НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Розробка мультиагентної бездротової Mesh – мережі Smart пристроїв

Виконав (-ла): студент (-ка) 4 курсу, групи ТІ-61

_____Остапюк Володимир Вікторович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Ковальчук А. М._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____

(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____доцент, к.т.н.Сірий О.А_____ _____

(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2020

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.В. Коваль

(підпис)

”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Остапюк Володимир Вікторович

(прізвище, ім’я, по батькові)

1. Тема роботи Розробка мультиагентної бездротової Mesh – мережі Smart пристроїв

керівник роботи Ковальчук Артем Михайлович, доцент

(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __

2. Строк подання студентом роботи 15 червня 2020

3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Ubuntu Linux, мови програмування C, Java, Dart

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналаізувати поточні програмно-апаратні комплекси, вибрати і обгрунтувати вибір програмно-апаратних засобів розробки, розробити архітектуру програмно-апаратного комплексу, розробити даний програмно-апаратний комплекс, зробити висновки за разультататами роботи

5. Перелік ілюстративного матеріалу

1.Цілі та задачі роботи 2. Вимоги до розроблюваної системи 3. Структура системи 4. Технології, використані при розробці комплексу 5. Структура 6LoWPAN мережі 6. 6LoWPAN мережа в IPv6 мережі. 7. Схема роботи серверного рівня 8. Діаграма класів сутності агента системи. 9. Базова діаграма прецедентів роботи з системою 10. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання "11" жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.2019	
2.	Вивчення та аналіз задачі	25.10.2019	

3.	Розробка архітектури та загальної структури системи	25.11.2019	
4.	Розробка структур окремих підсистем	12.12.2019	
5.	Програмна реалізація системи	10.03.2020	
6.	Оформлення пояснювальної записки	25.04.2020	
7.	Захист програмного продукту	10.05.2020	
8.	Передзахист	12.06.2020	
9.	Захист	15.06.2020	

Студент _____ Остапюк В.В.

(підпис) (прізвище та ініціали,)

Керівник роботи _____ Ковальчук А.М.

(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дана дипломна робота присвячена розробці бездротової мультиагентної системи. Предмет дослідження дипломної роботи – програмно-апаратний комплекс,, програмна частина якого на кінцевих пристроях написана на мові програмування C, а система збору даних та перегляду написана на мові програмування Java в середовищі IntelliJ IDEA, а мобільний застосунок написаний за допомогою мови програування Dart.

Розроблений програмно-апаратний комплекс забезпечує контроль та моніторинг над широким спектром параметрів навколишнього середовища та середовища проживання людини.

Записка містить 121 сторінок, 40 малюнків

Ключові слова: мультиагентна система, бездротова мережа, мікросервісна архітектура, Mesh-архітектура

SUMMARY

his business work is dedicated to manufacturers of wireless multiagent system. Thesis programs are software and hardware, the program of the part of end devices written in the C programming language, the data acquisition and viewing system is written in Java in the IntelliJ IDEA environment, and the mobile application is written to use the Dart language.

A software and hardware complex of controlling and monitoring a wide range of models of surrounding people and people has been developed.

The note contains 121 pages, 40 pictures

Keywords: multiagent system, wireless measure, microservice architecture, Mesh architecture

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TCP – Transmission Control Protocol

ADC – Analog Digital Converter

PWM – Pulse-Width Modulation

MCU – Microcontroller Unit

I2C – Inter-Integrated Circuit

UDP – User Datagram Protocol

UART – Universal Asynchronous Receiver/Transmitter

SPI – Serial Peripheral Interface

ВСТУП

На сьогоднішній день популярною є використання різноманітних систем пристроїв для контролю і моніторингу параметрів навколишнього середовища для забезпечення більш простої і адаптивної взаємодії людини з ним. Зараз існує велика кількість різноманітність інтелектуальних систем, що забезпечують моніторинг і контроль параметрів, що забезпечує адаптивне і безпечне середовище проживання для людини.

Одним із таких типів інтелектуальних систем є мультиагентні системи, що забезпечують адаптивне і масштабоване рішення для контролю і моніторингу над параметрами системи. Особливістю даного типу систем є те, що окремі пристрої системи є автономними і можуть вирішувати певну задачу контролю самостійно, але можуть і працювати в системі для вирішення певної комплексної задачі моніторингу. Тому для виявлення недоліків існуючих систем і розробки більш адаптивних рішень було прийнято рішення провести аналіз і розробити мультиагентну платформу

Для розробки даної мультиагентної платформи були використані мови програмування C, Java, Dart

1. ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ МУЛЬТИАГЕНТНОЇ БЕЗДРОТОВОЇ MESH – МЕРЕЖІ SMART ПРИСТРОЇВ

Сучасні системи контролю і моніторингу параметрів середовища проживання людини включають в себе певну кількість пристроїв-датчиків та пристроїв-контролерів, які на основі зібраних даних про навколишнє середовище змінюють їх відповідно до встановлених користувачем рекомендованих значень параметрів. Також одними з найпоширеніших пристроїв контролю і моніторингу є ті, що забезпечують безпечне функціонування критичних областей взаємодії людини з зовнішнім світом такі як датчики руху, датчики протікання та пристрої контролю і моніторингу за споживаного струму. Застосування даних систем безпеки забезпечує зменшення ризику таких небажаних процесів таких як проникнення зловмисників, затоплення будівлі при пошкодженні системи водопостачання чи необережності, не оптимальне використання електротехніки, що збільшує споживання електроенергії та збільшує ризик перевантаження системи. Використання даних систем забезпечує більш автоматизовану і адаптивну систему взаємодії людини з своїм середовищем проживання, що робить його більш безпечним, а також більш простим у взажмодії.

1.1 Аналіз існуючих систем

У процесі дослідження існуючих рішень в сфері контролю і моніторингу навколишнього середовища за допомогою мультиагентних систем було виявлено певну кількість проектів з даної тематики, які мають як спільні частини так і особливості.

“Система інтелектуальної безпеки AJAX” — це бездротова система безпеки будівлі розроблена українською компанією AJAX Systems. Система надає можливість контролю за безпекою будинку або квартири за допомогою різних

датчиків, що сповіщають про появу людей в певній області, небезпеку пожежі, протікання системи водозабезпечення. Основними перевагами даної системи є:

- використання передачі даних на частоті Sub-1GHz, що забезпечяує покриття великої площі через більшу максимально допустиму відстань (а саме 1500м) передачі ніж у мереж, побудованих на 2.4 GHz, що забезпечують максимальну відстань передачі у 100 м.
- Великий спектр пристроїв, що забезпечують різноманітний контроль та моніторинг за параметрами навколишнього середовища
- Якісна система шифрування повідомлень системи для забезпечення безпечного каналу зв’язку
- Робота пристроїв-датчиків на живленні від батарейки до 5 років
- Наявність пристроїв ретрансляторів, що збільшують покриття території

Недоліками цієї системи є такі особливості:

- система централізована, тобто при виході з ладу центрального пристрою - хабу, система не матиме змоги передавати повідомлення між пристроями і виконувати свої функції
- вибрана топологія мережі вимагає встановлення додаткових пристроїв ретрансляторів для забезпечення більшої площі покриття

“Система розумного будинку Clap” — це бездротова система розумного будинку, розроблена українською компанією Clap. Дана система схожа на попередньо зазначену, але пристрої даної системи функціонально більш націлені на контроль та

моніторинг параметрів проживання, а не на безпекові функції. Переваги даної системи є схожими до переваг попередньої системи, але є і кілька особливих моментів:

- наявність особливих датчиків, а саме пристрій контролю за рівнем опалення в приміщенні та інші
- інтеграція системи з системою оплати комунальних послуг в Україні на основі даних, зібраних датчиками даної системи

Недоліками даної системи є:

- топологія “зірка”, що обмежує максимальну площу покриття мережі до максимально можливої відстані передачі даних між двома вузлами мережі
- система централізована, що збільшує ризик втрати роботоспроможності системи у разі пошкодження головного пристрою системи

1.2 Вимоги до мультиагентної системи

Проаналізувавши вже наявні існуючі системи було визначені певні вимоги, якими повинна характеризуватися система і агенти системи, а саме:

- система має включати спектр пристроїв, що проводять аналіз та моніторинг основних параметрів навколишнього середовища
- пристрої системи мають бути розроблені за допомогою технологій, що забезпечують низький рівень енергоспоживання
- пристрої повинні бути частково або повністю автономними: при ймовірному зникненні мережевого зв’язку пристрої мають продовжувати збирати і аналізувати вимірювані параметри
- мережа система повинна бути побудована таким чином, щоб була можливість вільно масштабуватися в кількісному і територіальному змісті
- пристрої повинні мати можливість працювати за графіком для забезпечення зручного налаштування пристроїв системи

1.3 Мета та цілі дипломної роботи

Мета дипломної роботи — за допомогою розробленої мультиагентної системи покращити наявні технології контролю і моніторингу параметрів середовища проживання людини

Для досягнення цієї мети необхідно виконати такі завдання:

1. Провести аналіз необхідних характеристик, що визначають принципи роботи мультиагентної системи
2. Провести моделювання і проектування архітектури різних рівнів системи
3. Дослідити і проаналізувати засоби програмно-апаратної реалізації розробленої архітектури
4. Розробити даний програмно-апаратний комплекс

2. АНАЛІЗ ПРИНЦИПІВ РОБОТИ МУЛЬТИАГЕНТНИХ СИСТЕМ КОНТРОЛЮ ТА МОНІТОРИНГУ ЗА СЕРЕДОВИЩЕМ ПРОЖИВАННЯ

Кожна комплексна мультиагентна система контролю та моніторингу формується з певної кількості більш простих пристроїв, кожен з яких збирає дані про параметри певного типу і при потребі забезпечує можливість контролю даних параметрів.Ці різні пристрої мають спільний комплекс загальних параметрів, що змінює формат взаємодії з ним і вимоги, які до нього застосовуються. Ці загальні параметри включають до себе такі складові:

- набір параметрів навколишнього середовища чи певних процесів, що вимірюються
- набір параметрів навколишнього середовища чи певних процесів, що управляються
- набір конфігурацій, що задаються користувачем, що змінюють характер роботи системи:
    - частота вимірювання визначених параметрів, що контролюються даним пристроєм
    - певні граничні значення вимірюваних параметрів, що змінюють стан системи
    - набір реакцій,що виникають при зміні стану системи
- тип живлення , що визначає формування принципів алгоритму роботи пристрою
- робота в автономному чи мережевому режимі

Визначення цього набору параметрів визначає характер і алгоритм роботи пристроїв

мультиагентної системи і тип взаємодії цих пристроїв з іншими елементами системи і користувачем. Також задля забезпечення адаптивності і можливості розширення система має мати такі особливості:

- Визначений протокол мережевої взаємодії, що забезпечує надійність передачі повідомлення між вузлами мережі
- Визначена гарантована відстань передачі повідомлення між вузлами мережі
- Певний стандартизований формат повідомлень для взаємодії між різними вузлами мережі
- Визначений алгоритм приєднання нових пристроїв до працюючої мережі

3. ОПИС АРХІТЕКТУРИ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

Мультиагентна система буде складатися з різних пристроїв(агентів) та програмних комплексів, які розроблені незалежно і мають свої незалежні функції та взаємодіють за допомогою системи повідомлень. Основними компонентами системи є:

- Агент контролю та моніторингу поточної напруги і споживаного струму
- Агент контролю за освітленням
- Агент контролю наявності руху в просторі
- Агент збору даних про мікроклімат
- Агент контролю цілісності системи водопостачання
- Агент збору інформації про кількість відвідувань приміщення
- Агент реєстру і агент шлюзу до IOT-хмари
- Віддалена система контролю і моніторингу різних мультиагентних систем даного типу
- Мобільний застосунок - система, яка дозволяє отримувати інформацію про певну мультиагентну систему та змінювати її параметри з смартфону

Для кожного агента даної мультиагентної системи найважливішими характеристиками є їх функціонал, а також параметри роботи, що можуть бути змінені за допомогою користувацького інтерфейсу

Для функціонування і роботи даної мультиагентної системи розроблено декілька рівнів її функціонування, а саме:

- організація програмно-апаратного комплексу кожного агента
- організація мережевого рівня взаємодії агентів
- організація серверного рівня системи
- організація мобільного застосунку системи

3.1 Мережева архітектура системи

Комунікація між різними агентами в даній системі відбувається бездротово на радіочастотах менше 1GHz (Sub-1GHz), що збільшує відстань між пристроями, які обмінюються даними - до 1500 м.

Дана бездротова мережа є Mesh-мережею, перевагами яких є:

- Надзвичайно адаптивна та розширювана - елементи мережі можуть без будь-яких проблем вступати в мережу або виходити з неї, не порушуючи роботи мережі
- Підтримує велике покриття площі - при передаванні повідомлення між далекими членами мережі повідомлення буде відправлено по маршруту, який буде складатися із елементів, які є в зоні видимості кожного з елементів ланцюга передачі

Дана Mesh-мережа побудована по стандарту **6LoWPAN** (I*Pv6 over Low power Wireless Personal Area Networks*)[1], особливостями якого є:

- Ефективне використання IPv6 для бездротових бездротових мереж на простих вбудованих пристроях.
- Ідеально підходить для створення сітчастих мереж, він несе пакети даних IPv6 або v4 над IEEE 802.15.4 стандартом. Він забезпечує цільовий IP, при цьому забезпечує безперешкодне підключення до величезної кількості мереж,

використовуючи той самий стандарт, включаючи пряме підключення до Інтернету.

- Низькі швидкості передачі даних: Таким чином 6LoWPAN застосовує інший підхід до інших мережевих рішень бездротових датчиків низької потужності. Загальна система спрямована на забезпечення бездротового підключення до Інтернету при низькій швидкості передачі даних.
- Мала потужність: IPv6 забезпечує основний механізм транспортування для створення складних систем управління та економічного зв'язку з пристроями через бездротову мережу низької потужності.

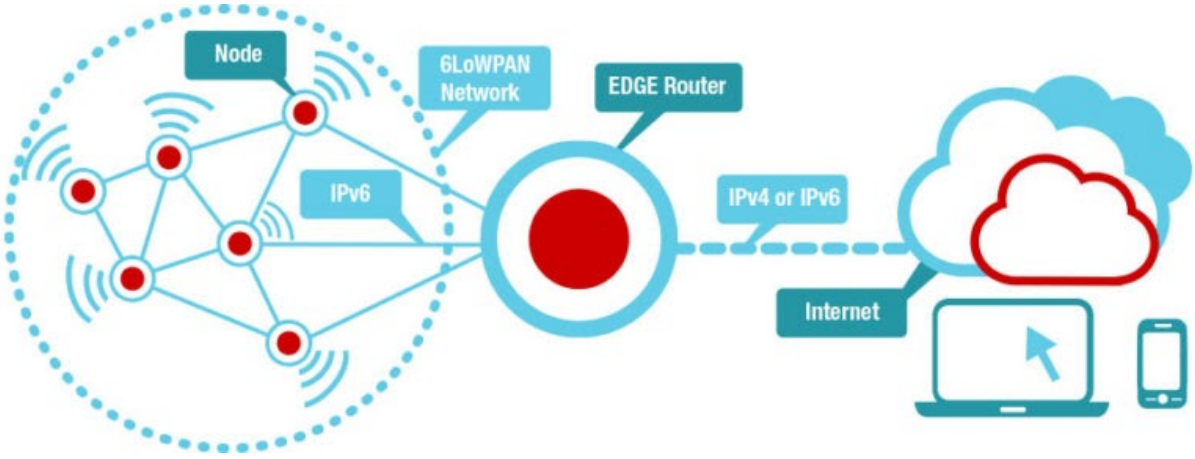

Рисунок 3.1 – мережева архітектура програмно-апаратного комплексу

Схему типової мережі побудованої по стандарту **6LoWPAN** можна побачити на рис. 3.1

Для захисту повідомлень всередині мережі використовується алгоритм шифрування AES-128

3.2 Архітектура серверного рівня системи

Серверний рівень мультиагентної системи має певний перелік вимог:

- Сумісність: Архітектура повинна повністю дозволяти спільно існувати неоднорідні технології. Рішення IoT швидко розвиваються; пристрої, протоколи, формати даних та інші технічні деталі досі розвиваються. Тому повинна бути можливість роботи різних частин системи з використанням різних алгоритмів і технології.
- Швидке розміщення нових части: Система повинна мати можливість доповнюватися додатковими модулями з повним циклом тестування і розробки (Agile / DevOps, CI / DC тощо) без зупинки чи перевантаження інших працюючих частин системи.
- Еластичність: система IoT матиме багато фізичних частин, розміщених у, можливо, жорстких умовах; будь-який окремий компонент повинен не збивати роботу всієї системи, що забезпечить більшу стійкість системи .
- Композиційність / повторне використання: Архітектура повинна забезпечувати використання різних послуг незалежно для створення більш складних функціональних можливостей системи. Маючи цю можливість, можна створити нові можливості системи, поєднуючи існуючі частини по-новому.Екосистема IoT може створити більшу цінність для клієнта, якщо основні послуги можуть бути використані багатьма зацікавленими сторонами для надання послуг більш високої вартості.
- Інтеграція: інтеграційний виклик IoT різноманітний і складний. Дана архітектура повинна підтримувати інкапсуляцію даних та логіки у «чорний ящик» , що доступний лише через інтерфейс, що не містить стану, що може спростити інтеграцію. Інтерфейс без стану легше інтегрувати, оскільки споживачеві послуги не потрібно буде

відстежувати переходи стану під час спілкування з постачальником послуг.

Архітектурний шаблон, що забезпечує виконання цього переліку вимог - архітектура мікро сервісів і саме ця архітектура використана для реалізації серверної частини програмно-апаратного комплексу мультиагентної системи.

Існує два варіанти побудови взаємодії різних сервісів у мікросервісній архітектурі:

- за допомогою безпосереднього доступу до інтерфейсу взаємодії з сервісом(наприклад REST), що робить різні сервіси жорстко зв’язаними, що зменшує надійність системи
- за допомогою брокера повідомлень, що доставляє різні повідомлення, що були створені різними сервісами, до сервісів, які підписані на обробку певного типу повідомлень

Тому для взаємодії між різними сервісами було використано технологію брокера повідомлень, а для взаємодії клієнтського застосунку з серверною частиною було використано технологію REST API.

3.3 Архітектура клієнтського застосунку

Мобільний застосунок розроблено за допомогою фреймворку, основою якого є реактивна зміна відображення даних, в залежності від поточного стану, а для використання і формування певної логіки обробки використовуються провайдери, моделі, різноманітні константи та інше (рисунок 3.2).

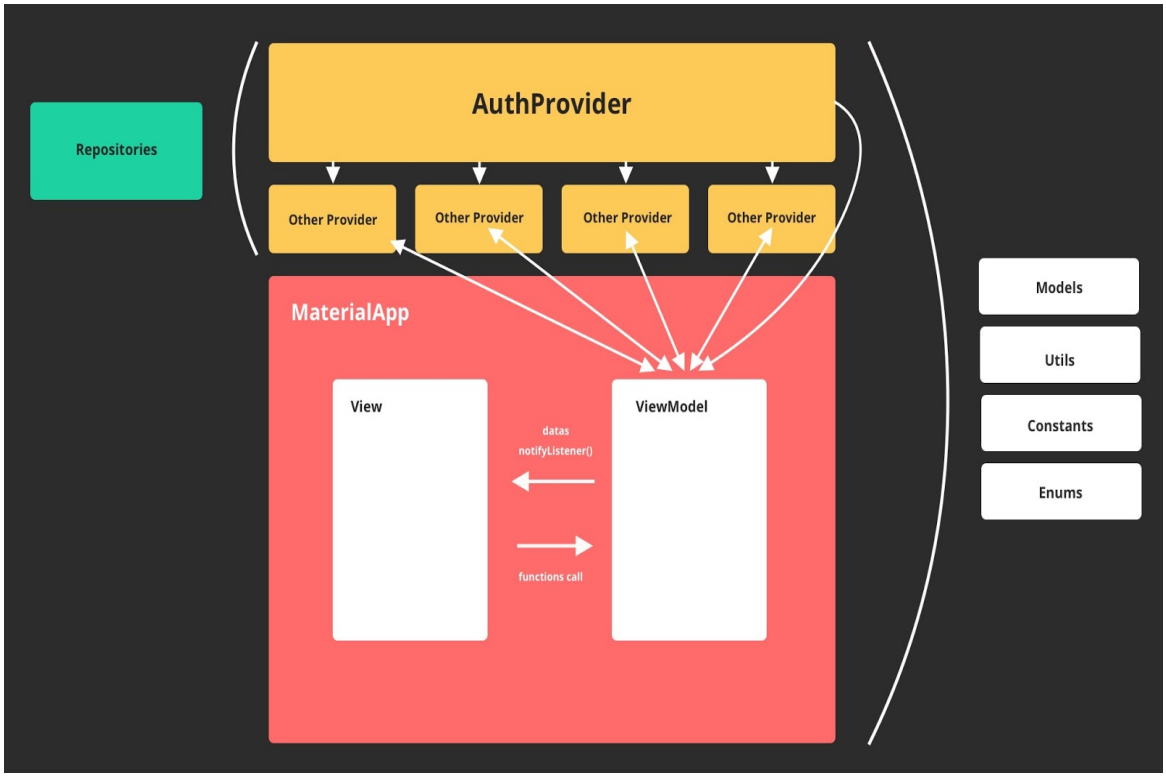

Рисунок 3.2 – Схема роботи додатку

Додаток ділиться на такі частини:

- Провайдер (Provider), включає в себе реалізацію певної бізнес-логіки застосунку.
- Модель представлення (ViewModel) забезпечує інтерфейс взаємодії Представлення з Провайдером.
- Представлення (View) забезпечує відображення даних і реакцію на певні дії користувача.

3.4 Обґрунтування вибору програмно-апаратної реалізації

При проектуванні системи було вивчено та проаналізовано предметну область та вимоги замовника. Після ретельного аналізу було вирішено розроблювати

програмний продукт, який заснований на інструментах, вказаних у вище зазначеному розділі

Основним мікроконтролером агентів даної системи вибрано мікроконтролер CC1310 фірми Texas Instruments, що має такі переваги:

Технології, які використовуються на серверному рівні системі, були обрані за принципом зручності у використанні, за те що є стандартом у побудові складних промислових системах, можливістю виконання на будь-якій операційній системі, можливістю працювати використовуючи мікросервісну архітектуру.Вибраний брокер повідомлень надає можливість створювати незалежні мікросервіси, які будуть містити свої частини бізнес-логіки і не залежати від інших мікросервісів

4. ОПИС ЗАСОБІВ ПРОГРАМНО-АПАРАТНОЇ РЕАЛІЗАЦІЇ

Програмно-апаратний комплекс був створений різними інструментами на кожному архітектурному рівні відповідно до тих вимог, яким повинна відповідати дана система.

4.1 Засоби програмно-апаратної реалізації агентного рівня системи

Для написання програмного коду всіх агентів було використано мову С та операційну систему реального часу TI-RTOS.

С - мова програмування високого рівня та загального призначення, яка ідеально підходить для розробки програмного забезпечення або портативних програм. Спочатку призначений для написання системного програмного забезпечення, С був розроблений в Bell Labs Денісом Річі для операційної системи Unix на початку 1970-х.

TI-RTOS [3] - це масштабована універсальна вбудована інструментальна екосистема для пристроїв TI. Він масштабується від багатозадачного ядра в реальному часі (SYS / BIOS) до повного рішення RTOS, включаючи додаткові компоненти проміжного програмного забезпечення та драйвери пристроїв. Забезпечуючи основні компоненти системного програмного забезпечення, які попередньо перевірені та інтегровані, TI-RTOS дозволяє зосередитись на створенні вашої програми.

У якості апаратної основа агентів було використано мікросхему CC1310 виробництва Texas Instruments. CC1310 - це пристрій сімейства CC13xx та CC26xx з економічно вигідними MCU з можливістю бездротової комунікації, здатними обробляти частоти 1 ГГц. Пристрій CC1310 поєднує в собі гнучку радіочастотну передачу дуже низької потужності з потужним 48-МГц Arm® Cortex®-M3 мікроконтролером на платформі, що підтримує декілька фізичних рівнів і стандартів радіочастотного зв’язку. Спеціалізований радіоконтролер (Cortex®-M0) обробляє

команди низькорівневого радіочастотного протоколу, які зберігаються в ПЗУ або ОЗУ, забезпечуючи таким чином наднизьку потужність і гнучкість. Низьке споживання енергії пристрою CC1310 не відбувається за рахунок продуктивності радіочастотного зв’язку; пристрій CC1310 відрізняється чудовою чутливістю та надійністю (селективністю та блокуванням)[2].

Мікроконтролер CC1310 має такі переваги:

- сучасна ARM Cortex-M3 архітектура
- багатий спектр модулів периферії - I2C,I2S,SPI,UART, 12-bit ADC
- апаратний модуль захисту даних по алгоритму AES-128,
- низьке енергоспоживання - в активному режимі: 2.5 mA, в режимі сну: 0.7 µA,
- режимі глибокого сну: 185nA
- модуль збору даних з аналогових датчиків з малим енергоспоживанням - одне пробудження щосекунди, виконуючи одне 12-бітне зчитування ADC: 0.95 µA,активний режим 24MHz: 0.4 mA + 8.2 µA/MHz
- можливість передачі даних по радіочастотах Sub-1GHz
- програмний стек для побудови мереж на основі частот Sub-1GHz
- переключення радіочастотних каналів зв’язку в рамках передачі даних між членами мережі для забезпечення бlльшої стійкості від неправомірного проникнення до мережі та глушіння

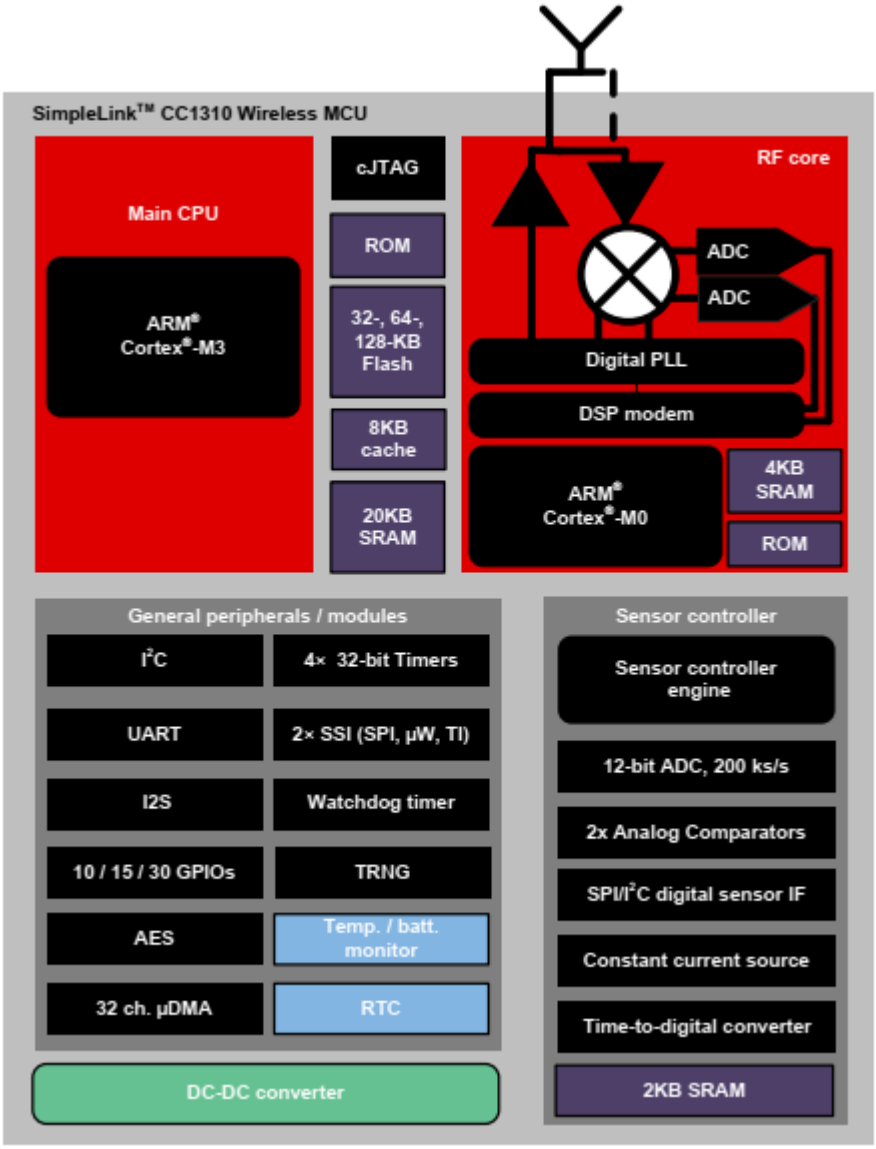

Рис 4.1. Функціональна схеми будови мікроконтролера CC1310

На рис. 4.1 можна побачити функціональну схему будови мікроконтролера CC1310

4.1.1 Засоби апаратної реалізації агента контролю та моніторингу поточної напруги і споживаного струму

Для побудови даного пристрою - агенту контролю та моніторингу поточної напруги і споживаного струму був обраний датчик споживаного струму виробника Allegro - ACS712, який має такі характеристики:

- Аналоговий шлях низького рівня шуму
- Час підйому на виході 5 мкс у відповідь на ступінчастий вхідний струм
- пропускна здатність 80 кГц
- Загальна похибка виходу 1,5% при TA = 25 ° C
- Невеликий корпус, низькопрофільний пакет SOIC8
- опір внутрішнього провідника 1,2 мОм
- 5,0 В, один режим живлення
- чутливість на виході від 66 до 185 мВ / А
- Вихідна напруга, пропорційна струмам змінного чи постійного струму
- Фабрично оброблені для точності
- Надзвичайно стабільна вихідна напруга зміщення
- магнітний гістерезис майже нульовий
- Радіометричний вихід від напруги живлення

Пристрій складається з точної, низькозміщеної, лінійної схеми датчика Холла з мідним кондуктором, розташованим біля поверхні матриці. Прикладений струм, що протікає через цей мідний шлях провідності, створює магнітне поле, яке відчувається
інтегрованим IC Холла і перетворюється на пропорційну напругу. Точність пристрою оптимізована завдяки близькій близькості магнітного сигналу до перетворювача Холла. Точну пропорційну напругу забезпечує низькокомплектний стабілізований мікросхема BiCMOS Hall IC, який запрограмований на точність після упаковки. Вихід пристрою має позитивний нахил (> VIOUT (Q)), коли збільшується

струм, що протікає через первинний шлях провідності міді (від пінів 1 і 2, до пінів 3 і 4), який є шлях, використовуваний для зондування струму.

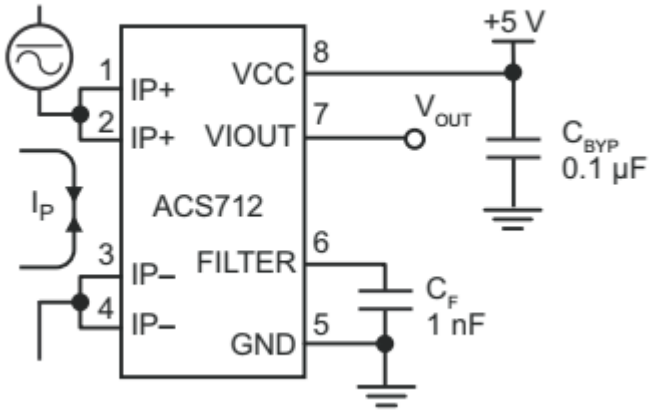

Рис 4.2 Типова схема підключення датчику ACS712

Товщина мідного провідника дозволяє вижити пристрою при до 5 × кратного перевантаження. Клеми провідного шляху електрично ізольовані від проводів датчика (піни 5 - 8). Це дозволяє використовувати датчик струму ACS712 у додатках, що вимагають електричної ізоляції без використання оптоізоляторів чи інших дорогих методів ізоляції. ACS712 поставляється в невеликому пакеті SOIC8 на поверхневому кріпленні. Рамка для свинцю покрита 100% матовим оловом, яке сумісне зі стандартними процесами складання друкованої плати без свинцю (Pb). Внутрішньо, пристрій не містить Pb, за винятком фліп-чіпних високотемпературних кульок припою на основі Pb, в даний час звільнених від RoHS. Пристрій повністю відкалібровано перед відправкою з заводу.

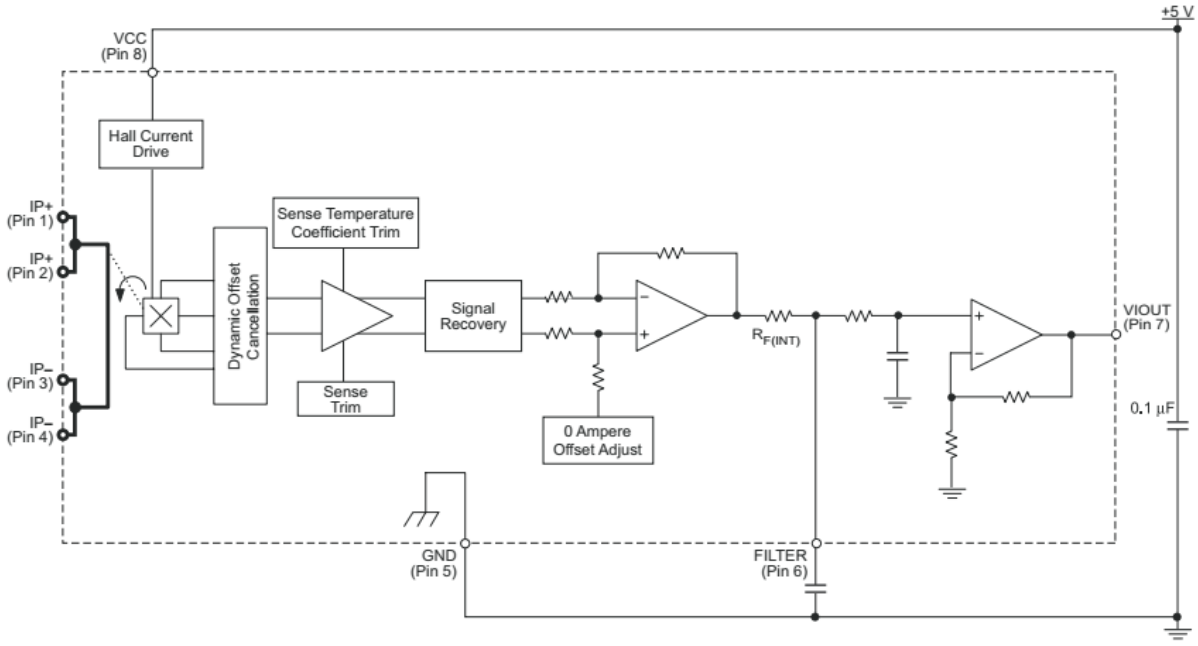

Рис 4.3. Функціональна схема датчику струму ACS712

4.1.2 Засоби апаратної реалізації агента контролю за освітленням

Для побудови даного пристрою - агенту контролю за освітленням був обраний датчик освітленості виробника Texas Instruments - OPT3002, який має такі характеристики:

- Широкий оптичний спектр: від 300 нм до 1000 нм
- Функція автоматичного повномасштабного налаштування спрощує програмне забезпечення та конфігурацію
- Рівні вимірювання: 1,2 нВт / см2 до 10 мВт / см2
- 23-бітний ефективний динамічний діапазон з автоматичним діапазоном посилення

- 12 двосторонніх, повномасштабних налаштувань діапазону: <0,2% (тип) відповідність між діапазонами
- Низький робочий струм: 1,8 мкА (тип)
- Робоча температура: від -40 ° C до + 85 ° C
- Широке джерело живлення: від 1,6 В до 3,6 В
- 5,5-В толерантного вводу / виводу
- Гнучка система переривань
- Малий форм-фактор: 2,0 мм × 2,0 мм × 0,65 мм

Світло-цифровий датчик OPT3002 забезпечує функціональність оптичного вимірювача потужності в межах одного пристрою. Цей оптичний датчик значно покращує продуктивність системи у порівнянні з фотодіодами та фоторезисторами. OPT3002 має широку спектральну пропускну здатність, що становить від 300 нм до 1000 нм. Вимірювання можна проводити від 1,2 нВт / см2 до 10 мВт / см2, без необхідності вручну вибирати повномасштабні діапазони за допомогою вбудованої повномасштабної функції налаштування. Ця можливість дозволяє вимірювати світло в 23-бітовому ефективному динамічному діапазоні. Результати компенсуються темновими струмами, а також іншими коливаннями температури. OPT3002 використовується в оптичних спектральних системах, які потребують виявлення різних довжин хвиль, таких як діагностичні системи на основі оптики. Система переривання може підсумовувати результат вимірювання одним цифровим виходом. Споживання енергії дуже низьке, що дозволяє OPT3002 використовуватись як датчик пробудження з низьким енергоспоживанням, що працює від акумулятора, коли закрита система доступна через певні інтерфейси доступу. OPT3002 повністю інтегрований і забезпечує оптичне зчитування живлення безпосередньо з сумісного, двохпровідного, послідовного інтерфейсу I2 і SMBus. Вимірювання проводяться як безперервно, так і однократно. Повна робоче енергоспоживання OPT3002 становить лише 0,8 мкВт при 0,8 вимірюваннях за секунду при потужності 1,8 В

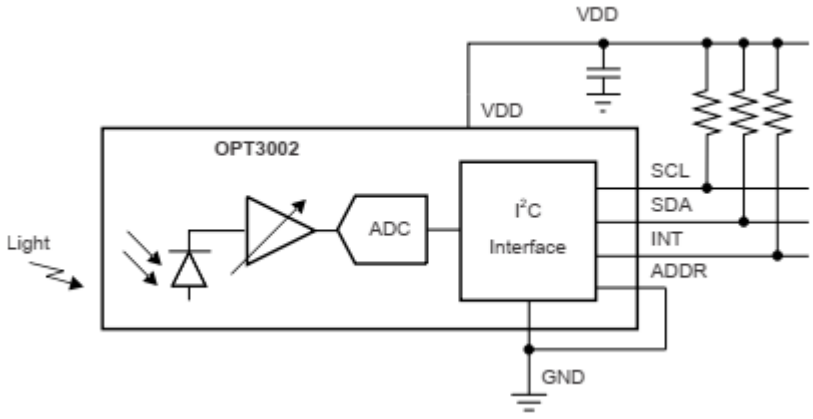

Рис.4.4 Функціональна схема побудови OPT3002

Перше вимірювання, яке пристрій проводить в режимі автоматичного діапазону, - це вимірювання діапазону 10 мс. Потім пристрій визначає відповідний повномасштабний діапазон для здійснення першого повного вимірювання. Для наступних вимірювань повномасштабний діапазон встановлюється результатом попереднього вимірювання. Якщо вимірювання спрямоване на нижню сторону повномасштабної, то повномасштабний діапазон зменшується на одну або дві настройки для наступного вимірювання. Якщо вимірювання спрямоване до верхньої сторони повномасштабної, то повномасштабний діапазон збільшується на одну установку для наступного вимірювання.

Якщо вимірювання перевищує повномасштабний діапазон внаслідок швидко зростаючої оптичної перехідної події, то поточне вимірювання перервано. Про це невірне вимірювання не повідомляється. Для оцінки та правильного скидання повномасштабного діапазону проводиться вимірювання 10 мс. Потім проводиться нове вимірювання з цим правильним повномасштабним діапазоном. Тому під час швидко зростаючого оптичного перехідного періоду в цьому режимі вимірювання, можливо, може зайняти більше часу
заповнити та повідомити, ніж вказано полем часу перетворення реєстру конфігурації (CT).

4.1.3 Засоби апаратної реалізації агента контролю наявності руху в просторі

Для реалізації функціоналу виявлення руху та вимірювання продовжуваності руху був обраний пасивний інфрачервоний датчик руху (PIR). Пасивний інфрачервоний датчик (PIR) вимірює інфрачервоне випромінювання, яке випромінюють теплогенеруючі об'єкти, а отже, інфрачервоне випромінювання у своєму полі зору. Кристалічний матеріал посередині прямокутника на поверхні датчика виявляє інфрачервоне випромінювання. Датчик розділений на дві половини, щоб він виявляв зміни ситуації, що виникають при входженні в поле об’єкта, а не випромінювання. Такі зміни кількості інфрачервоного випромінювання в елементі, у свою чергу, змінюють результуючу напругу, яка вимірюється бортовим підсилювачем. При виявленні руху датчик PIR посилає на вихідний контакт високий сигнал, який може бути прочитаний мікроконтролером або запущений транзистором для переходу на високий рівень.

Насправді було виявлено спотворене поле "нормальної" температури. Для запису зміни не потрібно розбивати поле об'єктом різної температури, оскільки датчики високої чутливості активуються лише під час руху. Призначений для використання при температурі від 15 ° C до 20 ° C, поле зору звужується при високих температурах, а якщо воно нижче 15 ° C, поле зору розширюється і дрібні або віддалені предмети можуть активувати датчик. З цієї причини не рекомендується дивитися на датчики в сухому середовищі, поблизу обладнання ОВК або у вікнах, які можуть спричинити зовнішню температуру або навіть рух.

Загалом, датчики PIR, які використовуються в системах освітлення та сигналізації в приміщенні, становлять близько 6 метрів, залежно від ситуації. Датчик адаптується до повільно мінливих умов, які зазвичай трапляються в навколишньому середовищі, але дає високі результати у разі раптової зміни.

Взагалі датчики PIR невеликі, недорогі, малої потужності, довговічні, мають широкий спектр лінз, прості в інтерфейсі та прості у користуванні. Їх найкраща особливість полягає в тому, що вони не зношуються. Вони можуть бути простими у використанні, але вони також дуже складні, оскільки існує багато змінних, які можуть змінювати вхід і вихід датчика.

Датчик PIR зазвичай має дві частини, кожна з яких зроблена з чутливого до інфрачервоного матеріалу матеріалу. Якщо вони не використовуються, обидві частини виявляють однакову кількість інфрачервоного випромінювання. Коли людина / тварина потрапляє у зону контролю датчика, половина з них зберігає інфрачервоне випромінювання, що спричиняє позитивну зміну між двома половинами. Відбудеться негативна диференціальна зміна після проходження об’єкта. Саме ці імпульси зміни визначаються датчиком PIR.

PIR-датчики поширені в пристроях детектора руху, які є пасивними інфрачервоними детекторами (PID), які називаються пасивними. У цьому випадку датчик PIR встановлений на друкованій платі, яка інтерпретує сигнали з мікросхеми піроелектричного датчика. Є два основні способи спрямування інфрачервоної енергії на поверхню датчика: (1) PID-вікно або кришка мають лінзи Френеля, які збирають та спрямовують світло з широкого огляду. Датчик PIR або (2) PID розділяє параболічні дзеркала, що фокусуються на інфрачервоній енергії всередині нього.

Через невеликий фізичний розмір сенсорних елементів об'єктив Френеля зазвичай розміщують перед датчиком PIR для розширення дальності, а також розширення поля зору шляхом множення та фокусування ІЧ-енергії на малі елементи датчика. Таким чином, форма та розмір об'єктива визначають загальний кут виявлення та зону огляду. Стиль лінзи, як правило, вибирається в залежності від застосування і вибору розміщення датчика в середовищі. Виходячи з цієї інформації, для найкращих результатів датчик повинен бути розміщений таким чином, щоб рух було впоперек датчика, а не прямо в датчик і подалі від джерел високого або

змінного тепла, таких як отвори змінного струму та лампи.

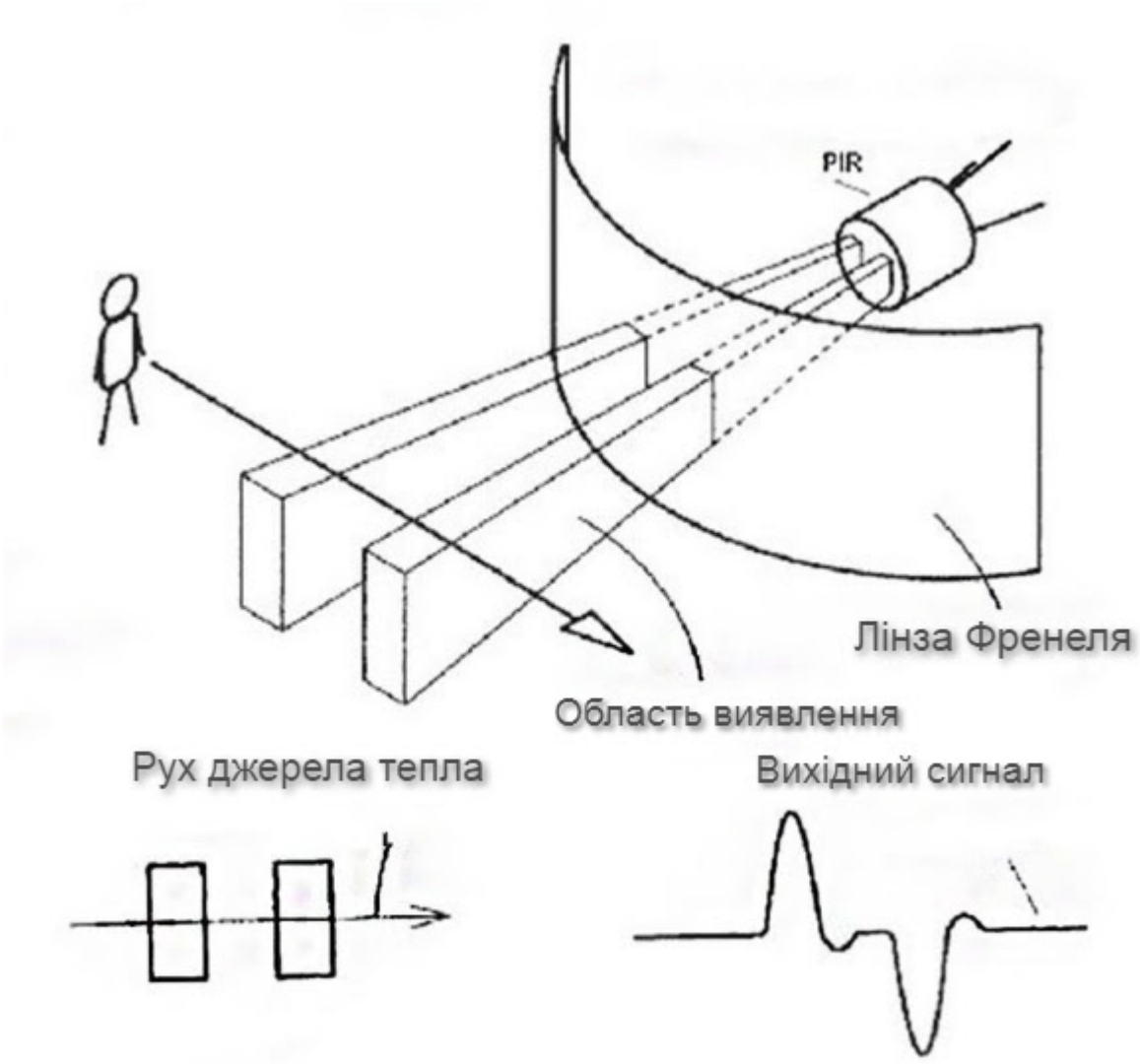

Рис 4.5. Принцип роботи PIR датчика

4.1.4 Засоби апаратної реалізації агента збору даних про мікроклімат

Для побудови даного пристрою - агенту збору даних про мікроклімат був обраний датчик температури і вологості виробника Texas Instruments - HDC1080, який має такі характеристики:

- Відносна точність вологості ± 2% (типово)
- Температурна точність ± 0,2 ° C (типова)
- Відмінна стійкість при високій вологості
- роздільна здатність вимірювання 14 біт

- Струм режиму сну 100 нА
- Середній струм подачі: калібрований.
- 710 нА @ 1sps, 11 біт Вимірювання RH
- 1,3 мкА @ 1sps, 11 біт RH і температури
- Напруга живлення від 2,7 до 5,5 В
- Невеликий слід пристрою 3 мм x 3 мм
- Інтерфейс I2C

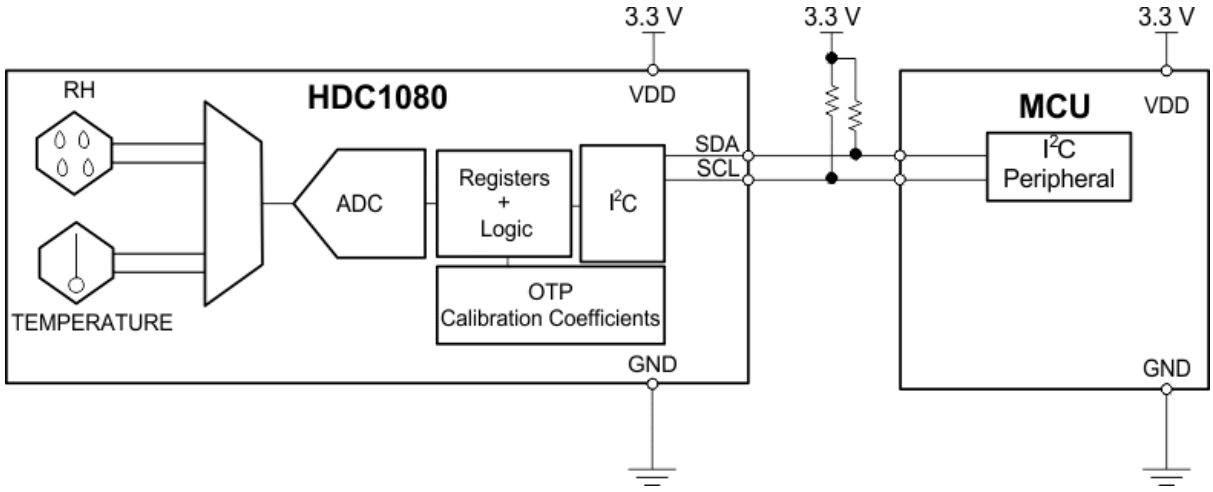

Рис.4.6 Функціональна схема датчику HDC1080

HDC1080 - цифровий датчик вологості з вбудованим датчиком температури, який забезпечує чудову точність вимірювання при дуже низькій потужності. Сенсорний елемент HDC1080 розміщений у верхній частині пристрою. Результати вимірювань можна зчитати через сумісний з I2C інтерфейс. Роздільна здатність базується на часі вимірювання і може становити 8, 11 або 14 біт для вологості; 11 або 14 біт для температури.

Однією з ключових особливостей HDC1080 є його низьке енергоспоживання, що робить пристрій придатним для використання в акумуляторі або в режимі енергозбереження. У цих додатках HDC1080 проводить більшу частину часу в режимі сну: при типовому 100 nA споживаного струму в режимі сну усереднене споживання струму мінімальне. Його низьке споживання в режимі вимірювання мінімізує будь-яке самонагрівання.

HDC1080 має два режими роботи: режим сну і режим вимірювання. Після ввімкнення живлення HDC1080 знаходиться в режимі сну. У цьому режимі HDC1080 чекає введення I2C, включаючи команди для налаштування часу перетворення, зчитування стану акумулятора, запуску вимірювання та зчитування вимірювань. Після того як він отримає команду для запуску вимірювання, HDC1080 переходить з сплячого режиму в режим вимірювання. Після завершення вимірювання HDC1080 повертається до сну.

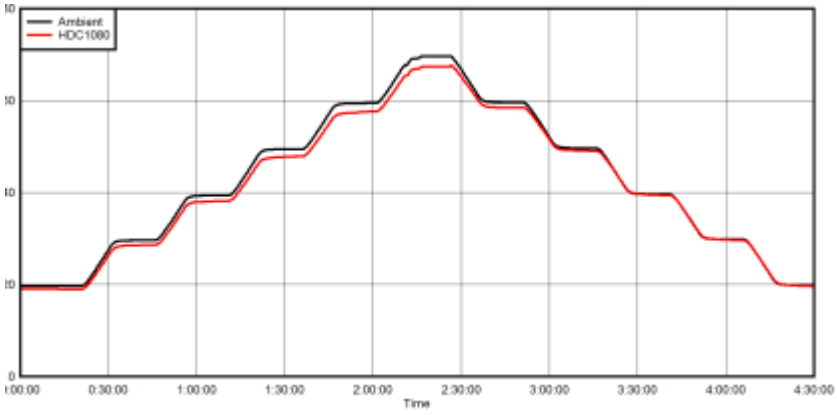

Рис.4.7 Співвідношення справжнього рівня вологості до виміряного

4.2 Засоби програмно-апаратної реалізації мережевого рівня системи

Для побудови мультиагентної мережі було використано мережевий стек TI15.4 для програмування мікроконтролерів CC1310, а в якості мережевого протоколу був вибраний протокол 6LoWPAN.

Даний стек має певні особливості, що вирізняють його серед інших мережевих рішень, одна з яких є режим стрибання частот (frequency hopping), що дозволяє збільшити стійкість системи до різноманітних шумів, створених іншими пристроями і мережами.

Програми, розроблені за допомогою стека, можуть бути налаштовані для роботи мережі в конфігурації частотних стрибків, коли мережеві пристрої скачуть на різних частотах.Стек TI 15.4 підтримує функцію нерозподіленого переходу каналів на основі режиму обміну кадрами (DFE) режиму специфікації Wi-SUN FAN v1.0[7].

Цією функцією можна керувати в режимі фіксованого каналу на будь-якому визначеному каналі або керувати нею в режимі стрибків каналів, коли послідовність стрибків каналу заснована на функції прямого хеш-каналу (DH1CF) . DH1CF генерує псевдовипадкову послідовність каналів, по яких переходити на основі розширеної адреси вузла; таким чином, псевдовипадкова послідовність каналів є унікальною для кожного вузла. Кожен вузол підтримує два типи послідовностей переходу каналів:

- Unicast
- Broadcast

Перестрибування частоти для кожного вузла базується на послідовності перестрибування каналу одноадресної передачі цих вузлів (див. Рис. 4.8.).

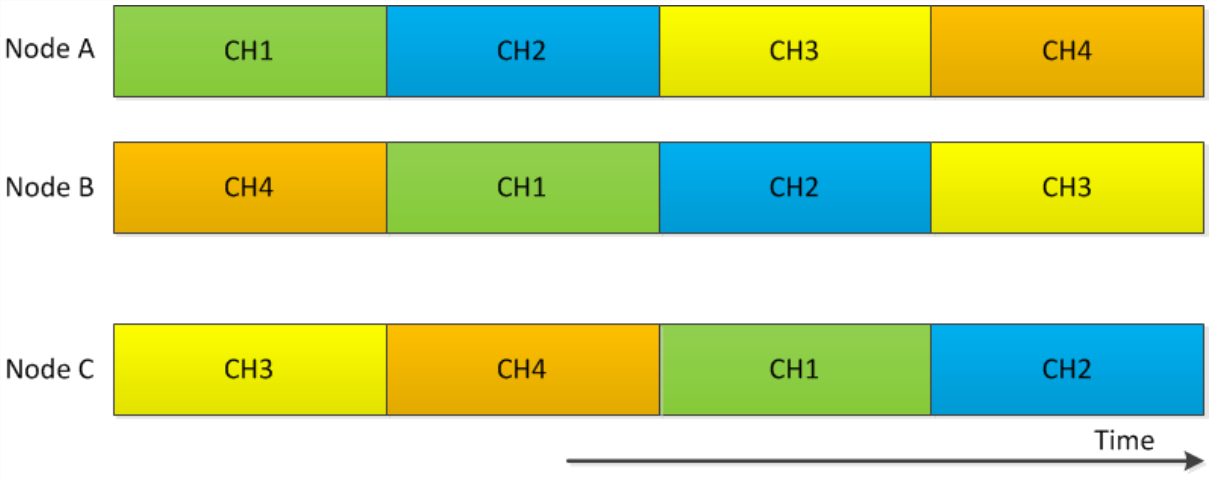

Рис. 4.8. Послідовність одночасного стрибка

Щоб увімкнути трансляцію трансляцій, координатор починає розклад мовлення (див. Рис. 4.9.). Кожен інший пристрій слідує послідовності трансляції, отриманій від координатора PAN. Пристрій виконує односмуговий стрибок до наступного часу перебування в ефірі. Потім пристрій перемикається на канал

переходу широкомовної передачі на час затримки трансляції і відновлює одноадресне стрибки в кінці інтервалу затримки трансляції.

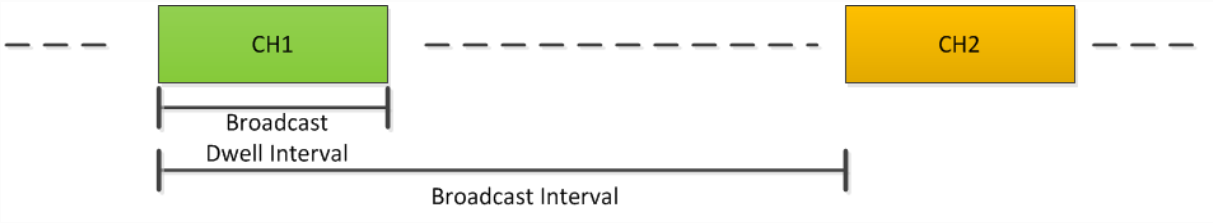

Рис 4.9. Послідовність стрибкового трансляції каналу

Додаток може вказати інтервал затримки мовлення (за замовчуванням - 250 мс), функцію переходу каналів (за замовчуванням - Фіксований канал) та список каналів для переходу (на основі ідентифікатора PHY Descriptor; наприклад, значення за замовчуванням 1 становить 129 каналів). Крім того, інтервал мовлення (за замовчуванням становить 4250 мс) може бути визначений координатором PAN. Коли для функції перескакування встановлено значення " *Виправлено"* , вузол повинен залишатися на каналі, встановленому ApiMac_FHAttribute_unicastFixedChannel або ApiMac_FHAttribute_broadcastFixedChannel під час одноадресної передачі та моменту часу трансляції відповідно.

Також підтримується спеціальний тип передачі, який називається *передачею* асинхронізації . У режимі стрибків частоти пристрій передає будь-який із типів кадру Async (як визначено в специфікації Wi-SUN FAN) у всіх запитуваних каналах . Це дає можливість стрибкового пристрою приймати такий кадр незалежно від послідовності стрибків. Таким чином, передача асинхронізації може використовуватися для обміну інформацією про перехід каналів. Ця функція особливо корисна при початковому формуванні мережі .

Коли приймається інформація про перехід каналу сусіда, TI 15.4-стек відслідковує стрибкові послідовності сусідніх стрибкових пристроїв і дозволяє успішно передавати одноадресні та широкомовні передачі; таким чином, приховуючи складності підтримки синхронізації від програми та полегшуючи завдання розробки додатків з функцією перестрибування частоти. Ключова відмінність від роботи в нечастотному режимі стрибків полягає в тому, що пристрої використовують в ефірі тільки розширену адресу (тобто **не** коротку адресу). За бажанням, коротка адреса може бути призначена на етапі асоціації, але не використовується для обміну даними.

4.2.1 **Операції в мережі TI15.4**

У режимі стрибків частоти вузли працюють як один із таких типів пристроїв:

- Координатор PAN
- Активний пристрій
- Сонний пристрій

Типова зіркова топологія має єдиний координатор PAN, підключений до набору неспаних або сонних пристроїв. Кожна мережа ідентифікується конкретним іменем мережі, що є 32 байтним значенням, а також 16-бітовим ідентифікатором PAN. Назва мережі (NetName) - це унікальний мережевий ідентифікатор, який налаштовується програмою, використовуючи атрибути частотного стрибків та атрибути бази інформації PAN (FH-PIB). Обслуговування NetName виходить за межі стеку MAC і не використовується стеком для фільтрації кадрів.

Розділи нижче пояснюють різні мережеві операції, важливі для розуміння мережі з підтримкою стрибків частоти

4.2.2 **Запуск мережі TI15.4**

На Рис 4.10. показано, як запускається частотна мережа при запуску координатора PAN в режимі частотного стрибка.

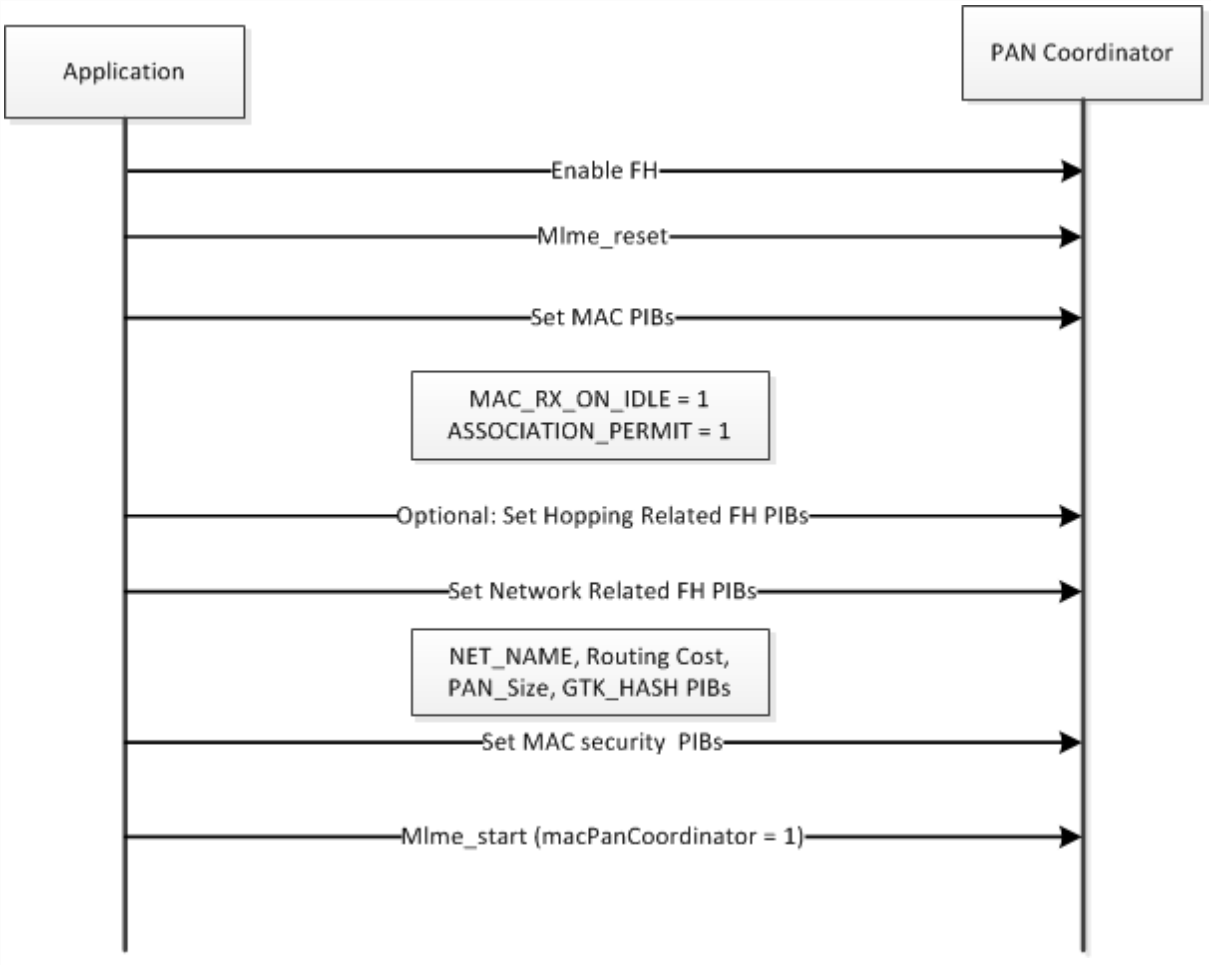

Рис 4.10. Послідовність запуску координатора PAN

Атрибути PIB, які відносяться до конфігурації стрибків частоти, пояснюються в конфігурації стека . NetName - це 32-розрядне значення ASCII, яке встановлюється програмою. Вартість маршрутизації повинна бути нульовою.

Спочатку розмір PAN повинен бути встановлений на нуль; пізніше розмір PAN повинен бути оновлений на основі кількості приєднаних вузлів.

4.2.3 **Запуск пристрою в стеку TI15.4**

На Рис 4.11. показана послідовність запуску пристрою.

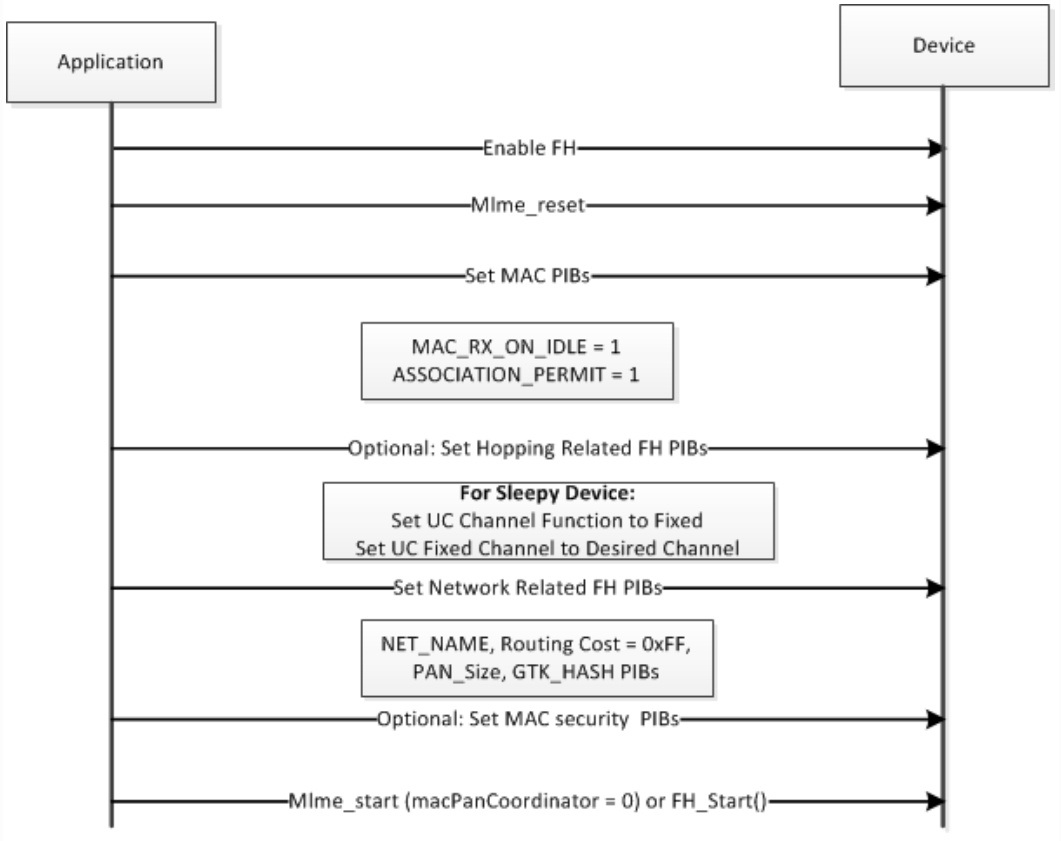

Рис 4.11. Послідовність запуску пристрою

Wi-SUN FAN v1.0 не визначає роботу режиму сну, але TI 15.4-стек реалізує патентоване розширення щодо поведінки, визначеної в протоколах Wi-SUN FAN та IEEE 802.15.4 MAC , щоб увімкнути сонний режим пристрої в мережах з частотним стрибком. Функція переходу каналу для сонного пристрою повинна бути встановлена на *Фіксований* , а нерухомий канал можна встановити на будь-який потрібний канал. Ключі безпеки можна встановити під час запуску (якщо ключі безпеки вже попередньо налаштовані для мережі), або ключі безпеки можна встановити після отримання їх за допомогою обміну ключами. Протокол обміну ключами повинен оброблятися над рівнем MAC.

Вартість маршрутизації повинна бути встановлена на високе значення (0xFFFF), щоб вказати, що пристрій не приєднався до мережі; пізніше це може бути оновлено програмою на основі використовуваного метрики маршрутизації.

Для приєднання до мережі вузол повинен пройти дві фази, описані нижче.

4.2.3.1. Фаза 1: Обмін інформацією про послідовність каналів через асинхронність повідомлення

Асинхронні повідомлення надсилаються назад до визначеного списку каналів. Ця дія дозволяє одержувачу приймати такі кадри з високою ймовірністю, незалежно від послідовності стрибків. Чотири різні асинхронні повідомлення підтримуються TI 15.4-стеком, як визначено в специфікації Wi-SUN FAN. Всі асинхронні кадри передаються на основі таймеру струменевого струму [RFC 6206]. Imin, Imax і K для алгоритму протікання рекомендується встановлювати на 1 хв, 16 хв та 1 відповідно.

Короткі описи чотирьох типів асинхронних повідомлень:

- PAN Advertisement solicit (PAS):
    - Повідомлення PAS використовуються пристроєм для запиту координатора або інших об'єднаних вузлів для передачі рамки рекламних повідомлень PAN.
    - Після отримання кадру PAN Advertisement, приєднується додаток може виявити IE NetName у кадрі, а потім використовувати ім'я, щоб визначити, чи слід скидати таймер струму ПА.
- PAN Advertisement (PA):
    - Кадри PA можуть передаватися координатором або об'єднаним вузлом для інформування сусідів про розмір PAN, вартість маршрутизації та ідентифікатор PAN.

    - Таймер, пов'язаний з передачею PA, запрограмований для скидання після прийому кадру PAS.
    - Після прийому кадру PAS вузли зв’язуються з передавачем кадрів PA(зверніть увагу, що стрибкова послідовність проводиться в кадрі PA).
    - Пристрій може вибрати один з вихідних вузлів кадру PA як ретрансляційний для виконання протоколу аутентифікації та безпечного обміну ключами , який повинен бути реалізований програмою, що працює над стеком. Приклади програм (колектор і датчик), включені в стек **, не** демонструють цю особливість.
- Запит на конфігурацію PAN (PCS):
    - Коли пристрій має ключі групового основного ключа (GMK), які використовуються в мережі, пристрій може запитувати передачу кадру конфігурації PAN.
    - Повідомлення PCS передаються вузлом на запит сусідів або координатору передають кадр PAN Config.
- Конфігурація PAN (PC):
    - Повідомлення PC передаються координатором або об'єднаним вузлом на основі таймеру, який повинен бути скинутий після отримання кадру PCS.
    - Кадри PC несуть послідовність трансляції і хеш-значення списку ключів GMK, які активно використовуються.
    - Після отримання кадру PC пристрій виявляє, що обмін каналами скачується.

Використовуючи конфігурацію стрибків частоти в топології зіркової мережі, TI рекомендує наступне:

- Координатор PAN передає кадри PA та PC на основі окремих таймерів.
- Пристрої передають кадри PAS та PCS з метою приєднання; потім пристрої призупиняють таймер після успішного з'єднання.
- Пристрої також повинні реалізувати механізм придушення для обмеження кількості переданих кадрів PAS та PCS.

4.2.3.2. Фаза 2: Процедура власної асоціації для інформування координатора мережі про приєднання (це необов'язковий крок)

Оскільки процедура приєднання частоти стрибків частоти (визначена специфікацією Wi-SUN) без підтвердження, координатор PAN не може виявити, чи пристрій успішно приєднався до мережі.

На додаток до повідомлення координатору про те, що пристрій успішно приєднався до мережі, необов'язковий механізм дозволяє координатору PAN виявити, якщо вузол приєднання є сонним або постійно включеним через поле про можливість повідомлення про запит про асоціацію, яке надіслане пристроєм до Координатор PAN. Цей додатковий механізм необхідний для програми координатора PAN для визначення наступного:

- Якщо програма повинна забудувати повідомлення, поки пристрій не запитається протягом певного налаштованого часу, якщо повідомлення є для сонного пристрою
- Якщо програма повинна надіслати повідомлення OTA, як тільки буде створено запит на передачу повідомлення

На рис. 4.12 та на рис. 4.13. показана процедура відповідно до сонних та несонних пристроїв.

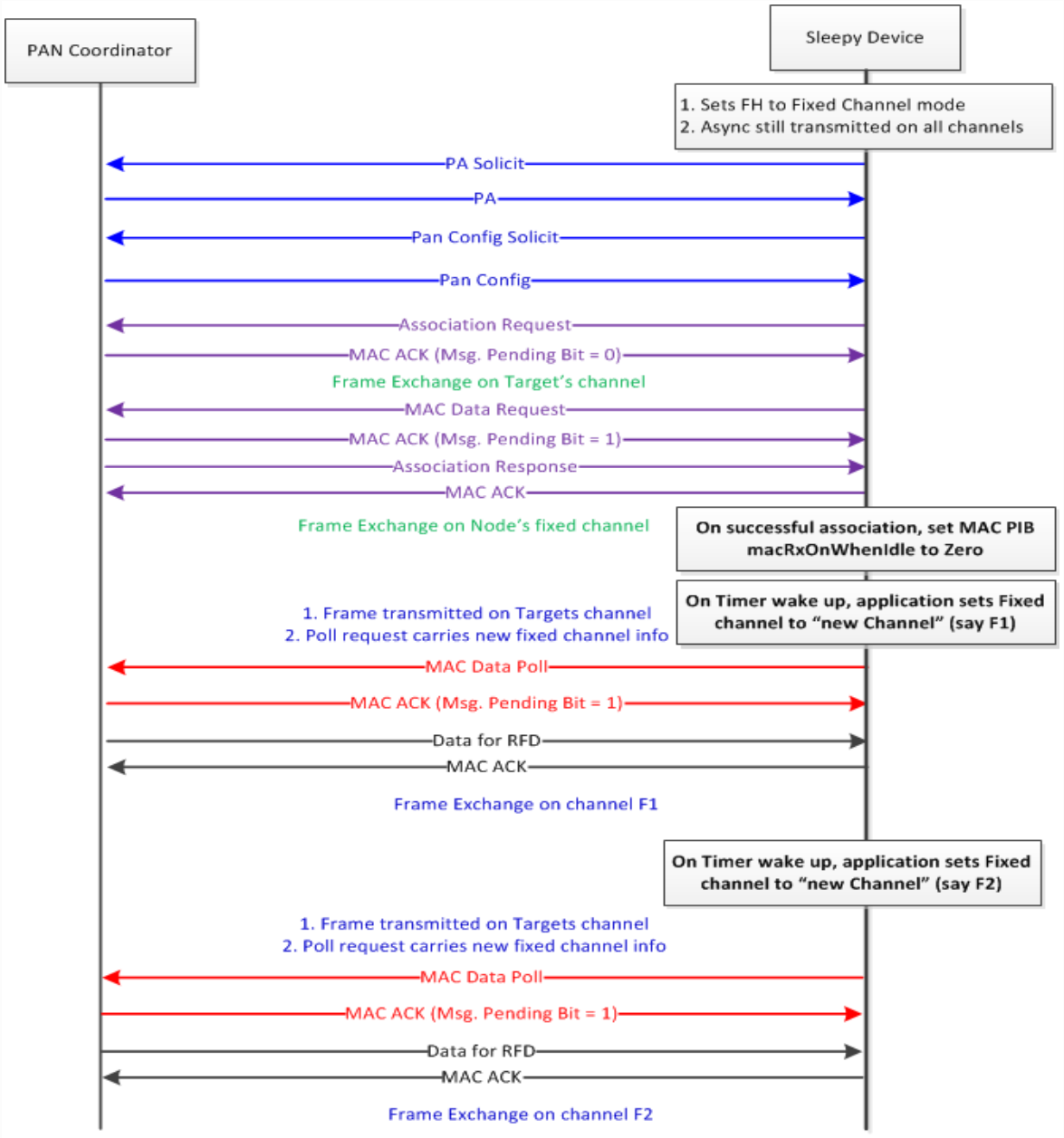

Рис. 4.12. Порядок приєднання до сонливого частотного пристрою

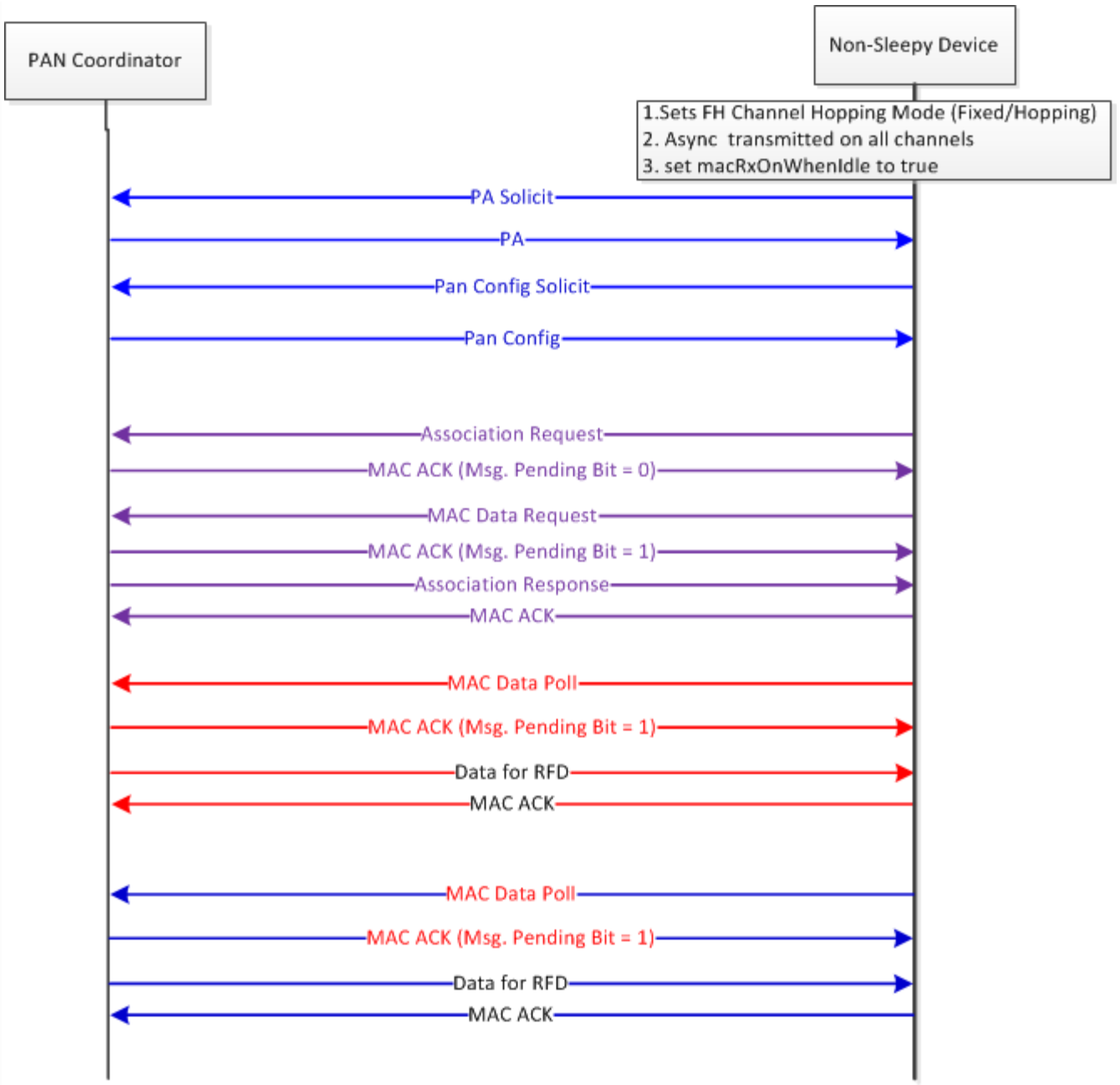

Рис. 4.13. Порядок приєднання для неслухаючого частотного пристрою

4.2.4 **Обмін даними в стеку TI15.4**

Три типи механізмів обміну даними підтримуються TI 15.4-стеком:

- Unicast-формат обміну даними
- Broadcast-формат обміну даними

- Асинхронний формат обміну кадрами

4.2.4.1. Обмін даними Unicast

Обмін одноадресною інформацією в режимі стрибків частоти відбувається на каналі вузла призначення. Вузол передає кадр на очікуваному приймальному каналі вузла призначення. Весь обмін кадрами відбувається на одному каналі (див. Рис. 4.14.). Подальший обмін даними відбувається на каналі, по якому приймач стрибає в момент передачі; цей подальший обмін даними не залежить від попередніх передач.

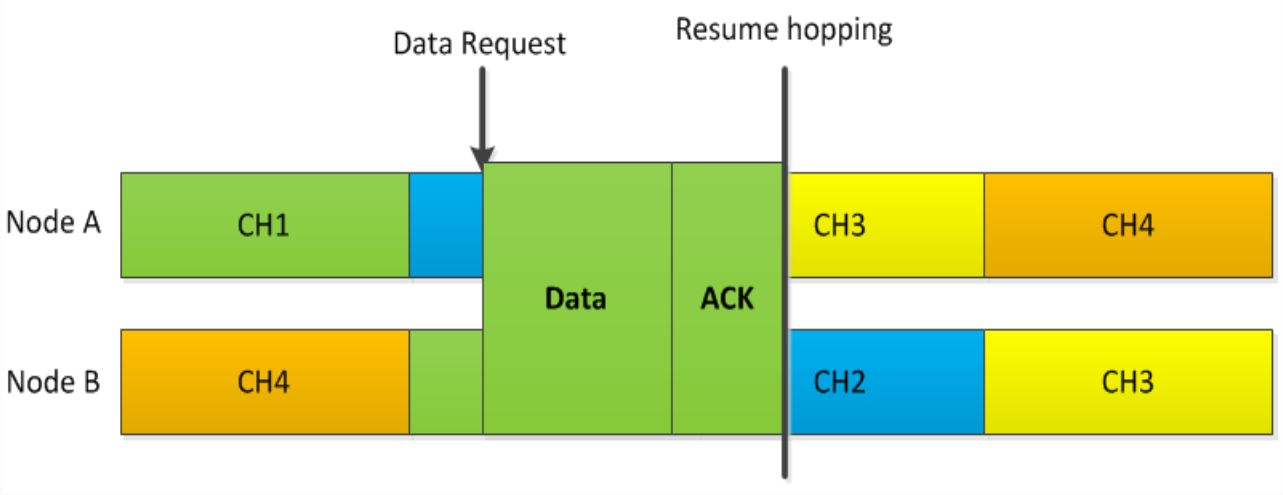

Рис 4.14. Обмін даними у конфігурації частотних переходів

Для передачі одноадресного кадру сусідові інформація про перестрибування вузол отримав у деякому попередньому кадрі. Інформація про перестрибування могла бути отримана через прийом будь-якого типу асинхронних кадрів з вузла призначення. Отже, програма повинна забезпечити отримання такого кадру від вузла призначення перед початком запиту даних.

4.2.4.2. Трансляція кадрів

Кадри широкомовної передачі передаються лише протягом інтервалу затримки трансляції . Кадри широкомовної передачі мають більш високий пріоритет, ніж кадри одноадресної передачі в такий час перебування, і вони виграють одноадресний кадр, коли він присутній. Ця різниця в пріоритеті може призвести до ситуацій, коли кадри вийшли з визначеного порядку(тобто порядок, коли кадри просять передати програмою, може відрізнятися від порядку, в якому вони передаються по ефіру).

Додаток на координаторі PAN може передавати кадр широкомовної передачі в будь-який час, оскільки координатор PAN починає стрибкову трансляцію, як тільки запускається програма. Всі інші вузли повинні дочекатися отримання кадру конфігурації PAN від координатора PAN для початку послідовності трансляції. Додаток на цих інших вузлах повинен чекати прийому кадрів конфігурації PAN; тоді додаток може встановити адресу джерела кадру конфігурації PAN (якщо він вирішить використовувати цей вузол як батьківський) , перш ніж надсилати запит на обмін кадрами широкомовної передачі.

4.2.4.3. Асинхронний обмін кадрами

Асинхронні кадри передаються пристроєм у списку каналів, визначених користувачем у запиті Async. На рис. 4.15. видно, що пристрій відхиляється від стрибкової послідовності і виконує цю операцію

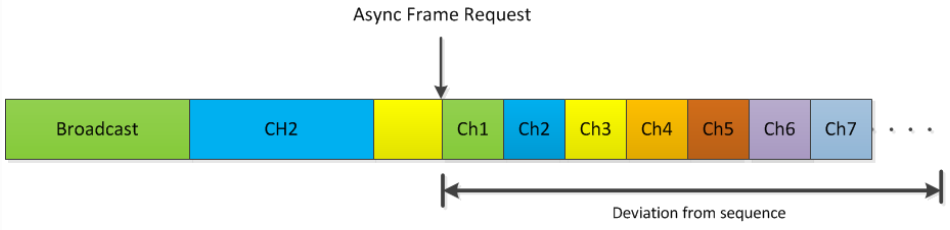

Рис 4.15. Асинхронний обмін кадрами

Завданням асинхронного обміну кадрами є передача даних на всіх доступних каналах (за замовчуванням = 129); таким чином, асинхронний обмін кадрами може зайняти кілька секунд (найгірший випадок - приблизно 4 секунди). Такі передачі, як правило, керуються таймерами, і їх не рекомендується часто передавати. За бажанням, пристрій може надіслати запит на асинхронізацію із командою Stop, яка зупинить поточний обмін кадрами Async.

4.2.5 Режим сну

Операція режиму сну пояснена на рис. 4.16. Оскільки процедура приєднання пояснюється , у цьому розділі наголошено на механізмі обміну даними.

При успішному об'єднанні MAC PIB macRxOnWhenIdle повинен бути встановлений на нуль, що дозволяє сонному пристрою вступати в роботу з малою потужністю. Сонний пристрій передає кадри координатору PAN на основі послідовності стрибків. Стек MAC на сонному пристрої працює на фіксованому каналі і не скаче самостійно.

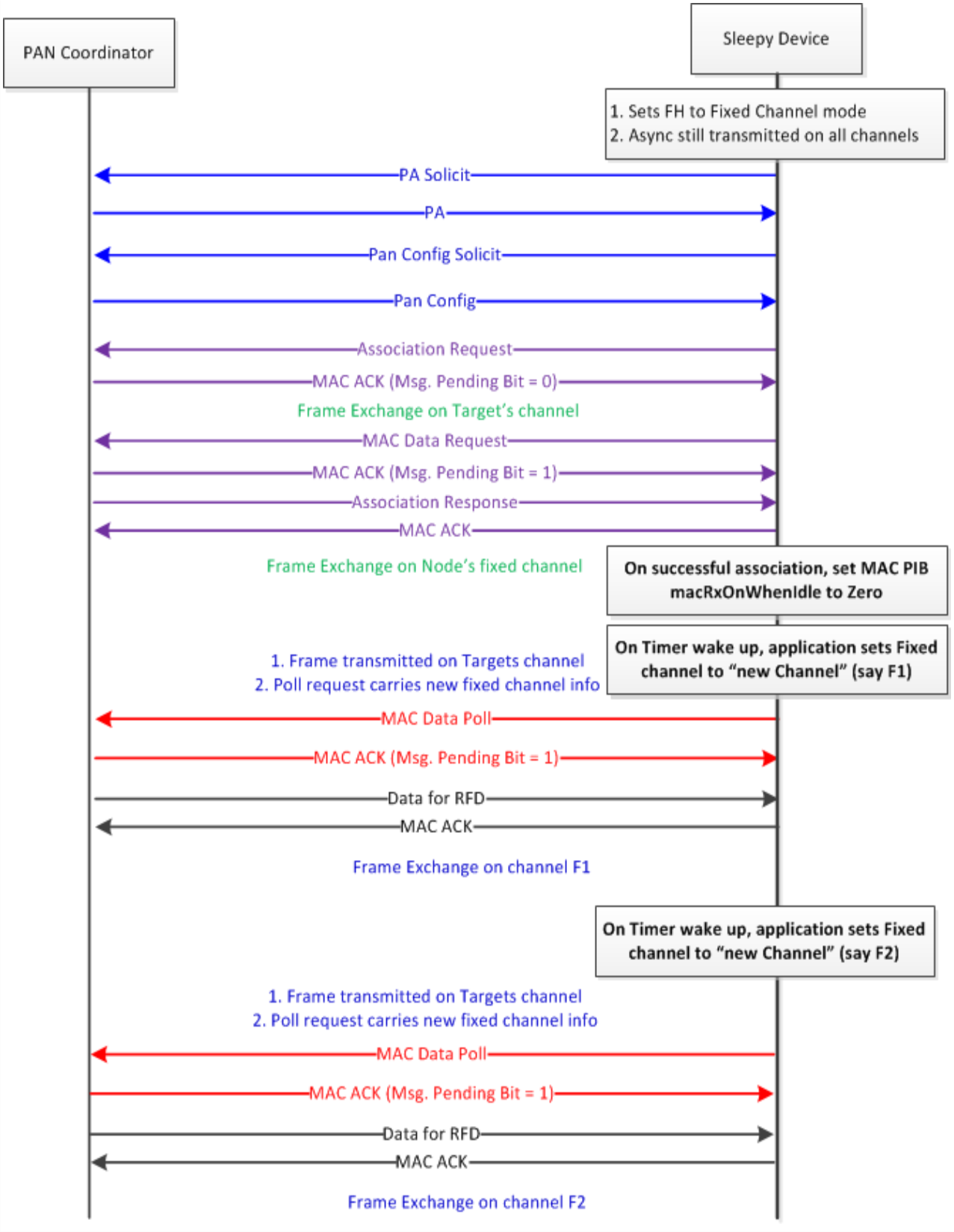

Рис. 4.16. Режим сну в режимі частотного переходу

Стек TI15.4 використаний в мультиагентній системі для побудови 6LoWPAN мережі.

6LoWPAN - це відкритий стандарт, визначений в RFC 6282 Інженерною робочою групою Інтернету (IETF), органом стандартів, який визначає багато відкритих стандартів, що використовуються в Інтернеті, такі як UDP, TCP та HTTP. Потужною особливістю 6LoWPAN є те, що, спочатку задуманий для підтримки бездротових мереж IEEE 802.15.4 з низькою потужністю в діапазоні 2,4 ГГц, він зараз адаптується та використовується для різноманітних інших мережевих носіїв, включаючи низьку потужність Sub-1 ГГц RF, Bluetooth® Smart, контроль лінії електропередач (PLC) та низькопотужний Wi-Fi®[8].

4.2.6 6LoWPAN мережева архітектура

На рисунку 4.17 показаний приклад мережі IPv6, що включає мережеву мережу 6LoWPAN. Точка доступу до Інтернету обробляється точкою доступу (AP), яка діє як маршрутизатор IPv6. Кілька різних пристроїв підключаються до AP в типових умовах, таких як ПК, сервери тощо. Мережа 6LoWPAN підключається до мережі IPv6 за допомогою крайового маршрутизатора. Крайовий маршрутизатор здійснює три дії:

1. обмін даними між пристроями 6LoWPAN та Інтернетом (або іншою мережею IPv6);
2. локальний обмін даними між пристроями всередині 6LoWPAN

3. створення та обслуговування радіомережі (мережа 6LoWPAN). Спільно спілкуючись з IP, мережі 6LoWPAN підключаються до інших мереж просто за допомогою IP-роутерів. Як показано на рисунку 4.18 , мережі 6LoWPAN, як правило, працюють на межі, виступаючи в ролі мереж. Це означає, що дані, що надходять у мережу, призначені для одного з пристроїв всередині 6LoWPAN. Одна мережа 6LoWPAN може бути з'єднана з іншими мережами IP через один або кілька крайових маршрутизаторів, які пересилають IP дейтаграми між різними носіями. Підключення до інших мереж IP може забезпечуватися через будь-яке довільне посилання, наприклад, Ethernet, Wi-Fi або 3G / 4G. Оскільки 6LoWPAN задає лише функцію IPv6 над IEEE 802.15.4 стандартом, крайові маршрутизатори можуть також підтримувати механізми переходу IPv6 для підключення мереж 6LoWPAN до мереж IPv4, таких як NAT64, визначених у RFC 6146. Ці механізми переходу IPv6 не потребують. Вузли 6LoWPAN для реалізації IPv4 в цілому або частково.

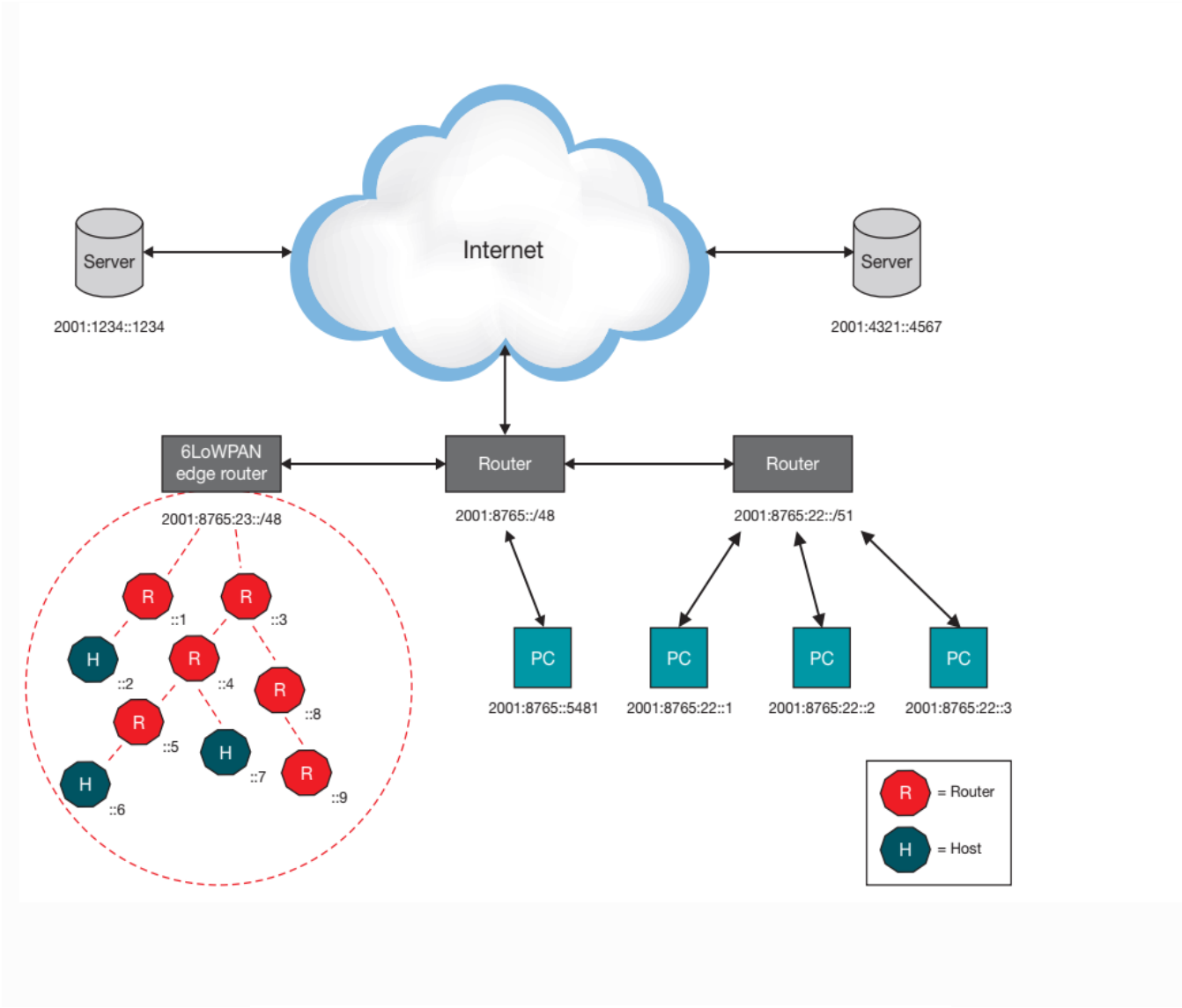

Рис.4.17 Приклад мережі IPv6 з сіткою 6LoWPAN

Оскільки крайові маршрутизатори пересилають дейтаграми на мережевий рівень, вони не підтримують жодного стану рівня додатків. Інші мережеві архітектури, такі як ZigBee®, Z-Wave, Bluetooth® або фірмові мережі, вимагають, щоб підключитися до мереж на базі IP-адреси, таких як Інтернет. Ці шлюзи додатків повинні розуміти будь-які профілі додатків, які можуть використовуватися в мережі, а будь-які зміни протоколів додатків на бездротових вузлах також повинні супроводжуватися змінами на шлюзі. На відміну від цього, маршрутизатори на основі IP-меж, як і крайовий маршрутизатор, залишаються агностичними для протоколів додатків, що використовуються в 6LoWPAN. Це знижує навантаження на

крайовий маршрутизатор з точки зору потужності процесора, завдяки чому можна використовувати вбудовані пристрої з меншою вартістю, більш просте програмне забезпечення та менш складне обладнання. Однак архітектура IP не виключає використання проксі-серверів та кешів для оптимізації продуктивності мережі, які сьогодні широко використовуються в Інтернеті. Два типи пристроїв включені в типову мережу 6LoWPAN: маршрутизатори та хости. Маршрутизатори можуть, як випливає з назви, маршрутизувати дані, призначені для іншого вузла в мережі 6LoWPAN. Хости також відомі як кінцеві пристрої і не в змозі направляти дані на інші пристрої в мережі. Хост також може бути сонним пристроєм, періодично прокидаючись, щоб перевірити його батьків (маршрутизатор) на наявність даних, що забезпечує дуже низьке енергоспоживання.

6LoWPAN докорінно змінює ландшафт IoT. Як і обговорювалося, до цього часу потрібен складний шлюз додатків для того, щоб пристрої, такі як ZigBee, Bluetooth та власні системи, підключалися до Інтернету. 6LoWPAN вирішує цю дилему, вводячи адаптаційний рівень між ланкою зв'язку стеку IP та мережевими рівнями, щоб забезпечити передачу дейтаграм IPv6 по радіосигналам IEEE 802.15.4. Усі системи зв'язку використовують набір правил або стандартів для форматування даних та контролю обміну. Найпоширенішою моделлю в системах передачі даних є модель Open Systems Interconnect (OSI), яка в спрощеній моделі розбиває комунікацію на п'ять фундаментальних шарів. На рисунку 4.18 показана ця спрощена модель OSI поряд із двома типовими прикладами стеків, що використовуються в пристроях IoT. Один - це пристрій, на якому працює стек Wi-Fi, інший - пристрій, підключений до IoT, на базі 6LoWPAN

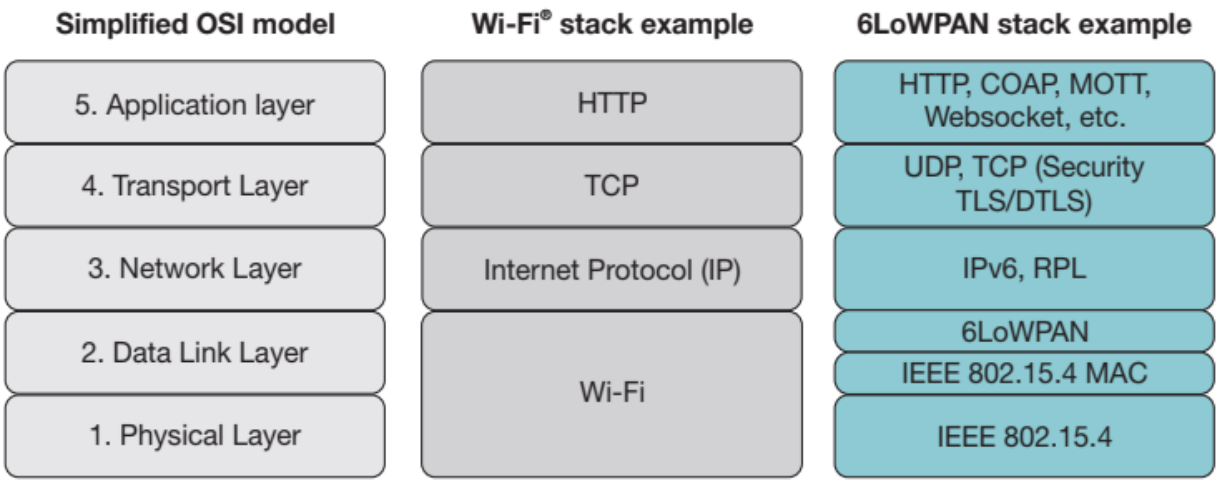

Рис. 4.18 Модель OSI, приклад стека Wi-Fi® та стек 6LoWPAN

Фізичний рівень перетворює біти даних у сигнали, які передаються та приймаються по повітрю. У прикладі 6LoWPAN використовується IEEE 802.15.4. Окрім чітко закріпленої версії стандарту 2006 року, існують дві важливі поправки: e та g. IEEE 802.15.4e - це поправка MAC та забезпечує вдосконалення, такі як стрибкоподібний сканування каналу (TSCH) та скоординоване прослуховування вибірки (CSL). Обидві вдосконалення мають на меті ще більше знизити енергоспоживання та зробити інтерфейс більш надійним. IEEE 802.15.4g є поправкою на PHY (або фізичний рівень) і має на меті забезпечити додатковий діапазон радіочастотних діапазонів, що дозволить використовувати у всьому світі навіть в смугах частот Sub-1 GHz. Рівень зв'язку даних забезпечує надійний зв’язок між двома безпосередньо підключеними вузлами шляхом виявлення та виправлення помилок, які можуть виникнути на фізичному рівні під час передачі та прийому. Рівень каналу передачі даних включає рівень доступу до засобів масової інформації (MAC), який забезпечує доступ до медіа, використовуючи такі функції, як багаторазовий доступ носія почуття - уникнення зіткнень (CSMA-CA), коли радіо слухає, що ніхто інший не передає, перш ніж фактично надсилати дані через повітря.

Цей шар також обробляє обрамлення даних. У прикладі 6LoWPAN, шар MAC - IEEE 802.15.4. Шар адаптації 6LoWPAN, що забезпечує адаптацію від IPv6 до IEEE 802.15.4, також знаходиться в шарі зв'язку. Мережевий рівень адреси та маршрутизації даних через мережу, якщо це потрібно протягом декількох стрибків. IP (або Інтернет-протокол) - мережевий протокол, який використовується для надання всім пристроям IP-адреси для транспортування пакетів з одного пристрою на інший.

Транспортний рівень генерує сеанси зв'язку між програмами, що працюють на кінцевих пристроях. Транспортний рівень дозволяє безлічі додатків на кожному пристрої мати власний канал зв'язку. TCP є домінуючим транспортним протоколом в Інтернеті. Однак TCP - це протокол на основі з'єднання (включаючи замовлення пакетів) з великими накладними витратами, а тому не завжди підходить для пристроїв, що вимагають наднизького енергоспоживання. Для таких типів систем UDP, з нижчими накладними витратами, протокол без підключення, може бути кращим варіантом. Приклади захищених транспортних шарів включають TLS (захист транспортного рівня), що працює на вершині TCP та DTLS, який базується на UDP. Нарешті, рівень програми відповідає за форматування даних. t також гарантує, що дані переносяться в оптимальних для програми схемах. Широко використовуваний прикладний рівень в Інтернеті - це HTTP, що працює над TCP. HTTP використовує XML, що є текстовою мовою з великими накладними витратами. Тому не оптимально використовувати HTTP у багатьох системах 6LoWPAN. Однак HTTP все ще може бути дуже корисним для зв'язку між 6LoWPAN та Інтернетом. З цієї причини, індустрія та спільнота розробили альтернативні протоколи прикладного рівня, такі як протокол обмеженого застосування (COAP), протокол повідомлень, що працює над UDP з бітовим оптимізованим механізмом REST, дуже схожим на HTTP. COAP визначений IETF в RFC 7252 і визначає

повторну передачу, підтверджувані та не підтверджувані повідомлення, підтримку сонних пристроїв, блокові передачі, підтримку передплати та пошук ресурсів. COAP також легко відображати на HTTP через проксі.

Ще один протокол рівня додатків, який слід згадати, - телеметричний транспорт черги повідомлень (MQTT), протокол з відкритим кодом, який був винайдений IBM. MQTT - це протокол публікації / підписки, що працює над TCP. Дані не транспортуються безпосередньо між кінцевими точками. Натомість брокер (тобто сервер) використовується для передачі повідомлень. MQTT представляє сутність "теми"; пристрої можуть публікувати та передплачувати різні теми. Після оновлення теми, на яку підписався певний пристрій, пристрій отримає сповіщення та отримає дані через брокера. Пристрої можуть використовувати подвійні знаки типу # і * для підписки на ієрархію тем. MQTT підтримує кілька рівнів якості обслуговування (QoS), забезпечуючи доставку повідомлень. Брокер може запускатись як локально, в IP-мережі, так і в Інтернеті, і в одній системі підтримується взаємодія декількох брокерів. Доступно декілька публічних брокерів, і багато постачальників хмарних послуг надають доступ до MQTT. Існує ще багато протоколів додаткового рівня, які можуть працювати через TCP / UDP. Перелічені тут спеціально націлені на програми IoT з низькою потужністю.

4.2.7 Internet Protocol version 6 (IPv6) з допомогою IEEE 802.15.4

Сьогоднішній Інтернет (і багато автономних IP-мереж) в основному базується на IPv4 і використовує 32-бітні адреси, що обмежує адресний простір до 4 294 967 296 унікальних адрес. Оскільки адреси були призначені користувачам (і пристроям), кількість непризначених адрес закономірно зменшилась. Виснаження IPv4-адреси відбулося 3 лютого 2011 року, хоча воно було значно затримане змінами адреси,

такими як переклад мережевих адрес (NAT). Це обмеження IPv4 стимулювало розвиток IPv6 у 90-х роках, який знаходиться у комерційному впровадженні з 2006 року. IPv6 охоплює адресний простір з 2128 та 3,4 * 1038 унікальних адрес. Цього має бути достатньо для масштабування Інтернету на наступні десятиліття - навіть з обіцянкою Інтернету речей, який, за оцінками, може включати 50 мільярдів підключених пристроїв до 2020 року.

Щоб визнати збільшення пропускної здатності, IPv6 збільшує мінімальний максимальний блок передачі (MTU) з 576 до 1280 байт. IPv6 також відображає зміни та вдосконалення технологій на рівні зв’язків, якими користується Інтернет. Ethernet є домінуючою технологією зв'язку, і її пропускна здатність роками зростала. Дзеркала Wi-Fi Можливості Ethernet підтримують MTU подібного розміру та дуже високу швидкість зв'язку. І Ethernet, і Wi-Fi працюють в умовах достатньої потужності та потужних пристроїв. З іншого боку, IEEE 802.15.4 був розроблений для обслуговування іншого ринку; довговічні програми, які потребують великої кількості низькозатратних пристроїв із надмірною потужністю. Пропускна здатність згідно з цим стандартом обмежена 250 кбіт / с, а довжина кадру обмежена 127 байтами, щоб забезпечити низькі частоти помилок пакетів і бітів у збитковому радіочастотному середовищі. Крім того, IEEE 802.15.4 використовує дві адреси: 16-бітну коротку адресу та розширену адресу EUI-64. Ці адреси зменшують накладні витрати та мінімізують потреби в пам'яті. Крім того, 6LoWPAN працює найчастіше на кількох стрибках, утворюючи мережеву мережу малої потужності, що принципово відрізняється від мереж на базі Ethernet або Wi-Fi. Нарешті, пристрої, що використовуються для реалізації 6LoWPAN, як правило, обмежені ресурсами, мають близько 16 кБ оперативної пам’яті та 128 кБ ПЗУ.

Через вищезазначені обмеження ресурсів та багатострибкову топологію 6LoWPAN, підтримка IPv6 через мережі IEEE 802.15.4 представляє кілька проблем;

1. Датаграми IPv6 не є природним придатним для мереж IEEE 802.15.4. Низька пропускна здатність, обмежена буферизація та дейтаграми, що становлять десяту частину мінімальної MTU IPv6, роблять необхідним стиснення заголовка та фрагментацію даних. Наприклад, заголовки посилань IEEE 802.15.4 можуть обмежити можливе корисне навантаження до 81 байт. Це робить заголовки IPv6 (40 байт), TCP (20 байт) та UDP (8 байт) занадто великими.

2. Оскільки IEEE 802.15.4 є протоколом як низько потужним, так і з низькою пропускною здатністю, крім використання RF як носія, він більш схильний до помилкових перешкод, збоїв у зв’язку та асиметричних зв’язків (A чує B, але B не чує A). Ці характеристики вимагають того, щоб мережевий шар був пристосованим та відповідальним одночасно з низькою потужністю та ефективністю.

3. Найпоширеніша мережа топологія для 6LoWPAN - це сітка з малою потужністю. Це заперечує припущення, що посилання - це єдиний домен широкомовної передачі, що є дуже важливим, оскільки саме фундамент IPv6, такий як відкриття сусіда, покладається на нього.

Вищезазначені питання розглядаються у стандарті 6LoWPAN.

Під час надсилання даних через шари MAC та PHY завжди використовується шар адаптації. Наприклад, RFC 2464 визначає, як пакет IPv6 інкапсульований у кадр Ethernet. Те ж саме використовується для IEEE 802.11 Wi-Fi. Для 6LoWPAN RFC 6282 визначає, як кадр даних IPv6 інкапсулюється через радіозв'язок IEEE 802.15.4.

Основна увага робочої групи IETF, 6LoWPAN WG, полягала в оптимізації передачі пакетів IPv6 по мережах низької потужності та втрат (LLN), таких як IEEE 802.15.4, і призвела до публікації уточнення RFC 6282;

- Стиснення заголовка, що стискає 40-байтні заголовки IPv6 та 8-байтові UDP, передбачаючи використання загальних полів. Поля заголовків видаляються, коли їх можна отримати із шару зв’язку. Спосіб стиснення заголовків є одним із факторів, що призвели до того, що стандарт підтримує лише IPv6, а не IPv4. Зауважте, що нічого не зупиняє запуск TCP в системі 6LoWPAN, але стиснення заголовка TCP не є частиною демістифікованого RFC 6282.6LoWPAN 7 жовтня 2014 року

- Фрагментація та повторна збірка. Посилання даних IEEE 802.15.4 з довжиною кадру не більше 127 байт не відповідає MTU IPv6, що становить 1280 байт. Слід зазначити, що формат кадру IEEE 802.15.4g не має однакових обмежень.

- Автоконфігурація без громадянства. Автоконфігурація без стану - це процес, коли пристрої в мережі 6LoWPAN автоматично генерують власну IPv6 адресу. Існують методи уникнення випадку, коли два пристрої отримують однакову адресу; це називається виявленням повторюваних адрес (DAD).

Протягом всього адаптаційного рівня 6LoWPAN ключовою концепцією є використання стиснення без стану або загального контексту для полегшення поля заголовка. Це може стиснути всі заголовки (адаптаційний, мережевий та транспортний шари) до кількох байт. Поле заголовків можна стиснути, оскільки вони часто несуть загальні значення. Загальні значення виникають через часте використання підмножини функціональності IPv6, а саме UDP, TCP та ICMP. Можна також зробити припущення щодо спільного контексту, наприклад, загального мережного префікса для всієї системи 6LoWPAN. Шар адаптації 6LoWPAN також видаляє дублюючу інформацію, яка може бути отримана з інших шарів, таких як адреси IPv6 та поля довжини UDP / IPv6.

4.2.8 Стиснення заголовка

Традиційний спосіб виконання стиснення заголовка IP - це статус, який використовується в з'єднаннях точка-точка, де потік між двома кінцевими точками стабільний. Ця реалізація є дуже ефективною в статичних мережах зі стабільними зв’язками. Комунікація через декілька стрибків вимагає перестрибування шляхом стиснення / декомпресії хопа. Протоколи маршрутизації (наприклад, RPL), як правило, що працюють в системах 6LoWPAN, отримують різноманітність приймача шляхом перенаправлення, що вимагатиме міграції стану і, отже, суттєво знижує ефективність стиснення. Для мереж, що динамічно змінюються, з безліччю стрибків і нечастою передачею, як радіомережа 6LoWPAN, слід застосувати інший метод. Замість цього в 6LoWPAN використовується стиснення без стану і загального контексту, яке не вимагає стану і дозволяє протоколам маршрутизації динамічно вибирати маршрути, не впливаючи на коефіцієнт стиснення.

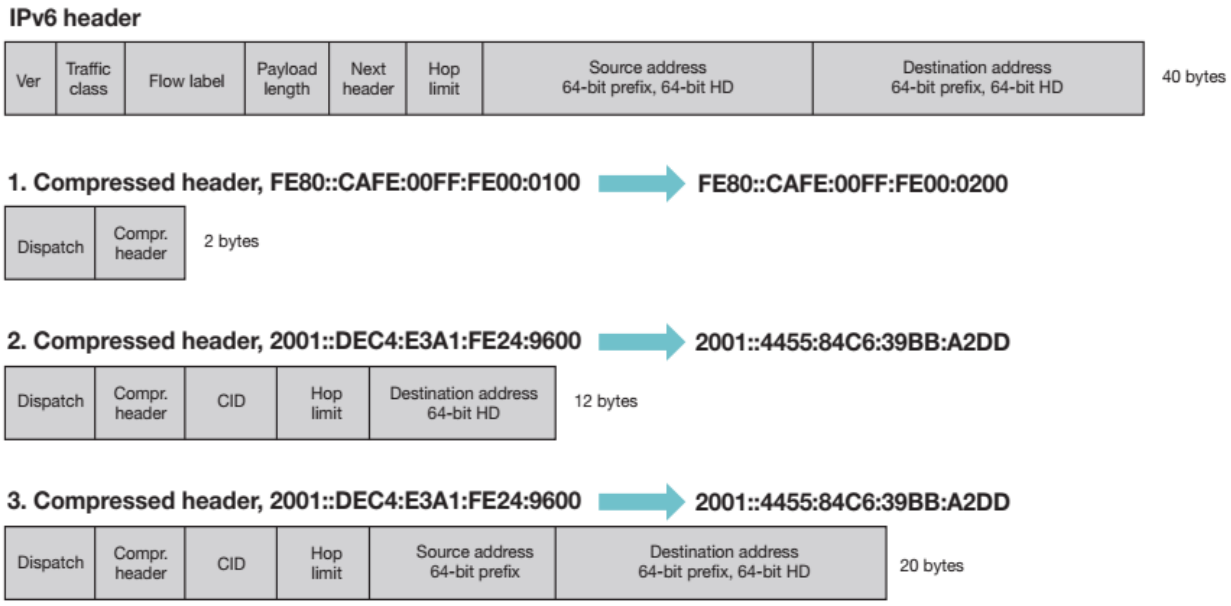

Рис. 4.19 Приклади стиснення заголовка 6LoWPAN IPv6

У прикладі на рисунку 4.19 відображаються три сценарії зв'язку:

1. Зв'язок між двома пристроями всередині однієї мережі 6LoWPAN, використовуючи локальні адреси посилань, заголовок IPv6 можна стиснути лише до 2 байтів.
2. Зв'язок, призначений для пристрою поза мережею 6LoWPAN, та префікс зовнішньої мережі відомий, де заголовок IPv6 може бути стиснутий до 12 байт

3. Подібно до 2, але не знаючи префіксу зовнішнього пристрою, що дає заголовку IPv6 у 20 байт.

Найкращий випадок (1) у цьому прикладі не корисний для надсилання даних додатків (оскільки він може використовуватися лише для надсилання даних прямим

сусідам), однак можливість стиснення заголовків даних, обмінюваних між двома найближчими пристроями, важливо, особливо для протокол маршрутизації. Найгірший випадок (3) все ще дає коефіцієнт стиснення 50 відсотків. У прикладі передбачається, що ідентифікатор інтерфейсу (IID) отриманий з MAC-адреси пристрою. Слід також зазначити, що стиснення заголовка UDP є частиною стандарту 6LoWPAN, як зазначено раніше в цьому документі, але не відображається в цьому прикладі.

4.2.9 Фрагментація та повторна збірка

Для того, щоб увімкнути передачу кадрів IPv6 по радіосигналах IEEE 802.15.4, кадри IPv6 потрібно розділити на кілька менших сегментів. Для цього генеруються додаткові дані в заголовках для збирання пакетів у правильній послідовності в кінці. Коли пакети даних повторно збираються, додана додаткова інформація видаляється, а пакети відновлюються до їх початкового формату IPv6. Послідовність фрагментації відрізняється залежно від типу маршрутизації (різні методи маршрутизації обговорюються пізніше). У випадку маршрутизації в мережі, фрагменти збираються лише в кінцевому пункті призначення, тоді як у випадку, коли маршрутизовані мережі пакети даних збираються повторно на кожному стрибку. Таким чином, у мережі з маршрутизатором кожен хоп повинен мати достатньо ресурсів для зберігання всіх фрагментів. В той час як в системі, що працює в сітці, багато мережевого трафіку генерується швидко, оскільки всі фрагменти передаються негайно. Якщо під час повторного збирання відсутні будь-які фрагменти (у системі з-під сітки), повний пакет потрібно повторно передавати. По можливості слід уникати фрагментації якомога довше, оскільки це негативно позначається на ресурсі акумулятора пристрою. Отже, максимальне значення має збереження корисної

навантаження (включає вибір відповідних протоколів рівня додатків) та використання стиснення заголовка.

4.2.10 Формати заголовків

6LoWPAN використовує складені заголовки та, аналогічні IPv6, заголовки розширень. Заголовки 6LoWPAN визначають можливість кожного підзаголовка. Визначено три підзаголовки: адресація сітки, фрагментація та стиснення заголовка.

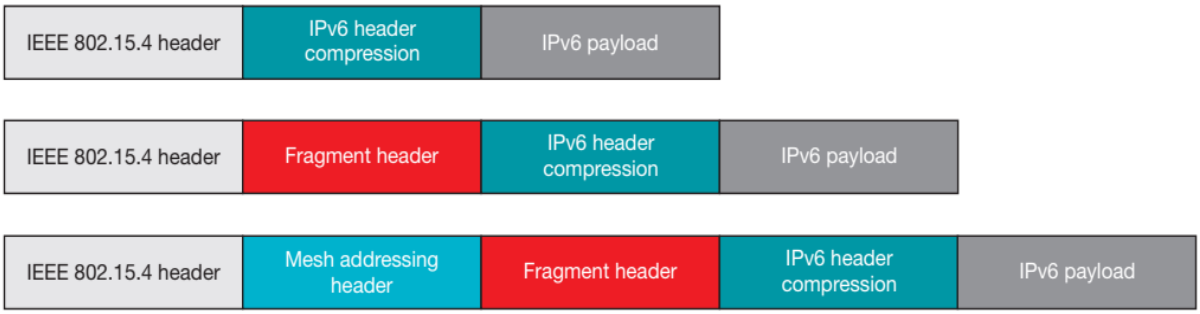

Рис. 4.20 Складені заголовки 6LoWPAN

Мережева адресація підтримує пересилку другого рівня (посилання на дані) та фрагментацію підтримує передачу IPv6 MTU. Формат заголовка визначається за допомогою поля типу заголовка, розміщеного на початку кожного заголовка. Стек заголовка легко аналізувати і дозволяє видалити підзаголовки, якщо це не потрібно. Заголовок фрагментації промальовується для пакетів, які вміщуються в один єдиний IEEE 802.15.4 кадр. Заголовок сітки не використовується під час надсилання даних лише через один скачок. Заголовок фрагмента використовується, коли корисна навантаження занадто велика, щоб вміститися в один IEEE 802.15.4 кадр. Заголовок фрагмента містить три поля; розмір дейтаграми, тег дейтаграми та зміщення дейтаграми. Розмір дейтаграми описує загальну (не фрагментовану) корисну навантаження. Тег дейтаграми ідентифікує набір фрагментів і використовується для

узгодження фрагментів одного і того ж корисного навантаження. Зсув дейтаграми ідентифікує зміщення фрагмента в межах не фрагментованого корисного навантаження. Довжина заголовка фрагмента становить 4 байти для першого заголовка і 5 байт для всіх наступних заголовків. Заголовок сіткової адреси використовується для пересилання пакетів декількох стрибків всередині мережі 6LoWPAN. Заголовок сіткової адреси включає три поля: ліміт переходу, адреса джерела та адреса призначення. Поле обмеження стрибків використовується для обмеження кількості переходів для переадресації. Поле зменшується при кожному стрибку. Як тільки кількість досягне нуля, пакет випадає. Поля адреси джерела та місця призначення вказують кінцеві точки IP-адреси. Обидва є адресами IEEE 802.15.4 і можуть бути короткими або розширеними, як визначено в стандарті IEEE 802.15.4. Довжина заголовка сіткової адреси становить від 5 до 17 байт, залежно від використовуваного режиму адресації.

4.2.11 Маршрутизація

Маршрутизація - це можливість передавати пакет даних з одного пристрою на інший пристрій, іноді за допомогою декількох стрибків. Залежно від того, на якому шарі розташований механізм маршрутизації, визначаються дві категорії маршрутизації: mesh-under або route-over.

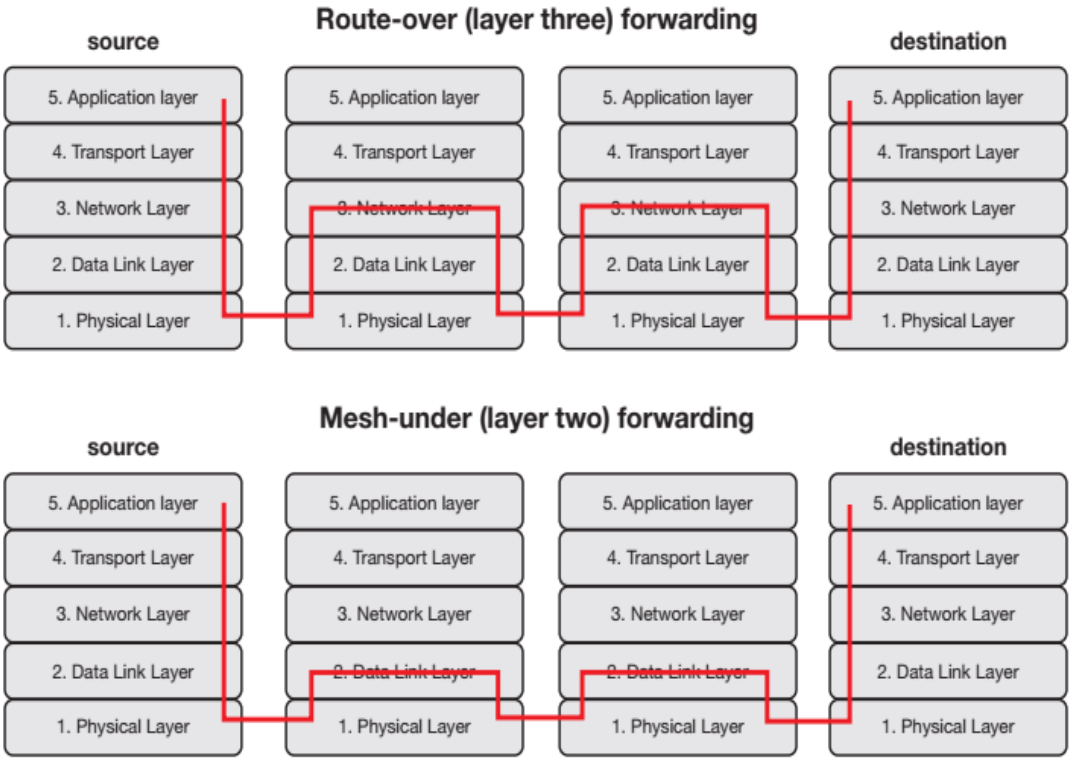

Рис. 4.21 Переадресація пакетів за сіткою та маршрутом

Mesh-under використовує адреси другого рівня (шар шару) (IEEE 802.15.4 MAC або коротка адреса) для пересилання пакетів даних; в той час як маршрутизація використовує три рівні (мережевий рівень) адреси (IP адреси). У системі mesh-under маршрутизація даних відбувається прозоро, отже, мережі mesh-under вважаються однією IP-підмережею. Єдиний IP-роутер у такій системі - крайовий маршрутизатор. Один широкомовний домен створений для забезпечення сумісності з протоколами IPv6 вищого рівня, такими як виявлення дублікатів адреси. Ці повідомлення потрібно надсилати на всі пристрої в мережі, що призводить до великого навантаження в мережі. Мережеві мережі найкраще підходять для менших і локальних мереж. У мережах маршрутизації маршрутизація відбувається на рівні IP, як описано вище, тому кожен перехід у таких мережах представляє один маршрутизатор IP. Використання маршрутизації IP забезпечує основу для більш

великих та потужних та масштабованих мереж, оскільки кожен маршрутизатор повинен реалізовувати всі функції, підтримувані звичайним IP-маршрутизатором, таким як DAD, тощо. Найпоширеніший протокол маршрутизації для мереж 6LoWPAN на сьогодні RPL (вимовляється "пульсація"), визначений IETF в RFC 6550. Порівняно з mesh-under, маршрутизація має перевагу в тому, що більшість протоколів, які використовуються на стандартному стеку TCP / IP сьогодні, можуть бути реалізовані та використані як є. RFC 6550 визначає протокол маршрутизації IPv6 для мереж з низькою потужністю і втратами (RPL), який забезпечує механізм, за допомогою якого багатоточковий трафік від пристроїв всередині мережі 6LoWPAN спрямований до центрального пункту управління (наприклад, сервера в Інтернеті) як а також підтримується точково-багатоточковий трафік від центральної точки управління до пристроїв всередині 6LoPWAN. Також доступна підтримка трафіку "точка-точка". Однак RPL не є оптимальним вибором для такого трафіку, оскільки дані у багатьох випадках потрібно переносити через крайовий маршрутизатор. RPL підтримує два різних режими маршрутизації; режим зберігання та режим незбереження. У режимі зберігання всі пристрої в мережі 6LoWPAN, налаштовані як маршрутизатори, підтримують таблицю маршрутизації та сусідню таблицю. Таблиця маршрутів використовується для пошуку маршрутів до пристроїв, а таблиця сусідів використовується для відстеження прямих сусідів вузла. У режимі зберігання єдиним пристроєм із таблицею маршрутизації є крайовий маршрутизатор, отже, використовується маршрутизація джерела. Маршрутизація джерела означає, що пакет включає повний маршрут (або стрибки), який йому потрібно пройти, щоб досягти пункту призначення. Наприклад, при передачі даних з одного пристрою на інший пристрій всередині тієї ж мережі 6LoWPAN дані спочатку надсилаються з вихідного пристрою на крайовий маршрутизатор, крайовий маршрутизатор, у свою чергу, здійснює пошук у своїй таблиці маршрутів і додає повний маршрут до призначення в пакеті. Режим зберігання пред'являє більш високі вимоги до пристроїв, що діють як маршрутизатори (тобто вони повинні мати

достатньо ресурсів для зберігання маршрутних та сусідніх таблиць), при використанні режиму, що не зберігає, накладні витрати збільшуються зі збільшенням кількості переходів, до яких потрібно пройти пакет, щоб досягти пункт призначення.

4.2.12 Автоматичне налаштування та пошук сусіда

Автоматична конфігурація - це автономне генерування IPv6 адреси пристрою. Процес істотно відрізняється між IPv4 та IPv6. У IPv6 він дозволяє пристрою автоматично генерувати свою адресу IPv6 без будь-якої зовнішньої взаємодії з сервером DHCP або подібним. Щоб отримати адресу, хост може спілкуватися через протокол виявлення сусідів (NDP), проте багато функцій NDP також включені в RPL. Описана тут процедура є дійсною і для RPL і включає чотири типи повідомлень:

- Прохання маршрутизатора (RS)
- Реклама маршрутизатора (RA)
- Суспільне прохання (NS)
- Суспільна реклама (NA)

IPv6 сусідське відкриття (ND) дозволяє пристрою виявити сусідів, підтримують інформацію про доступність, налаштовують маршрути за замовчуванням та розповсюджують конфігурацію 6LoWPAN, демістифіковані параметри 11 жовтня 2014 року. Повідомлення RS включає, серед іншого, префікс IPv6 мережі. Усі маршрутизатори в мережі періодично надсилають ці повідомлення. Якщо хост хоче брати участь у мережі 6LoWPAN, він присвоює собі локальну одноадресову посилання (FE80 :: IID), після чого надсилає цю адресу в повідомленні NS всім іншим учасникам підмережі, щоб перевірити, чи використовується адреса. кимось іншим. Якщо воно не чує повідомлення NA протягом визначених часових

рамок, воно передбачає, що адреса є унікальною. Ця процедура називається виявленням дублікатів адреси, DAD. Тепер, щоб отримати мережевий префікс, хост надсилає маршрутизатору повідомлення RS, щоб отримати правильний префікс. Використовуючи ці чотири повідомлення, хост може призначити собі унікальну в усьому світі IPv6 адресу. Використовуючи автоматичну конфігурацію адреси джерела, кожен хост генерує локальну IPv6-адресу посилання, використовуючи свою IEEE 802.15.4 EUI-64 адресу, 16-бітну коротку адресу або обидва. У конфігурації, що перебуває в сітці, сфера локальних зв'язків охоплює всю мережу 6LoWPAN, навіть через декілька стрибків, і локальна адреса посилання достатня для зв'язку, що відбувається в 6LoWPAN. Єдиний час, коли потрібна маршрутизована IPv6-адреса - це під час спілкування поза мережею 6LoWPAN. У конфігурації маршруту, локальна адреса посилання є достатньою для зв'язку з вузлами, які знаходяться в межах радіоохоплення, але потрібна маршрутизована адреса для зв'язку з пристроями, що знаходяться в декількох стрибках. Для всіх одноадресних адрес найефективніше отримати їх з локальної адреси IEEE EUI-64. Зв'язування 6LoWPAN між заголовками каналів зв'язку, адаптації та IP дозволяє усувати їх та усуває потребу в роздільній здатності адреси, що призводить до менших заголовків. Аналогічно, автоматична конфігурація повинна налаштувати адресацію інтерфейсу для використання загального префікса, щоб 6LoWPAN міг схилити префікс. 6LoWPAN може використовувати коротку адресу посилання для отримання адреси IPv6, що призводить до скорочення заголовків.

4.3 Засоби програмної реалізації серверного рівня системи

Для реалізації серверного рівня системи була використано мову програмування Java, фреймворк Spring, його реалізацію для мікросервісної архітектури Spring Cloud, а також Apache Kafka як брокер повідомлень.

Java - це мова програмування загального призначення, заснована на класах, об'єктно-орієнтована і покликана мати якомога менше залежностей від реалізації. Він призначений для того, щоб розробники додатків писали один раз, а програма працювала в будь-якому місці (принцип WORA), тобто компільований код Java може працювати на всіх платформах, які підтримують Java без необхідності перекомпіляції. Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від основної архітектури комп'ютера.

Термін "Spring"[4] означає різні речі в різних контекстах. Він може бути використаний для позначення самого проекту Spring Framework, саме з якого і почалося все. З часом інші Spring проекти були побудовані на основі Spring Framework. Найчастіше, коли люди кажуть «Spring», вони мають на увазі всю сім’ю проектів.

Spring Framework поділений на модулі. Програми можуть вибирати, які модулі їм потрібні. В основі лежать модулі основного контейнера, включаючи модель конфігурації та механізм введення залежності. Крім цього, Spring Framework надає основоположну підтримку різних архітектур прикладних програм, включаючи обмін повідомленнями, транзакційні дані та постійність та Інтернет. Він також включає серверну веб-структуру Spring MVC на основі сервлетів і, паралельно, реактивну веб-структуру Spring WebFlux.

Spring Cloud[5] пропонує інструменти для розробників для швидкого побудови деяких поширених моделей в розподілених системах (наприклад, управління конфігурацією, виявлення служби, автоматичні вимикачі, інтелектуальна маршрутизація, мікропроксі, шина управління, одноразові маркери, глобальні блокування, вибори керівництва, розподілені сеанси, стан кластерів). Координація розподілених систем призводить до шаблонів розробки, і за допомогою Spring Cloud можна швидко вставити служби та програми, які реалізують ці схеми. Вони добре працюватимуть у будь-якому розподіленому середовищі, включаючи власний

ноутбук розробника, центри обробки даних та керовані платформи, такі як Cloud Foundry.

Apache Kafka®[6] - це розподілена потокова платформа Потокова платформа має три ключові можливості:

- Публікувати та підписуватись на потоки записів, подібні до черги повідомлень чи системи обміну повідомленнями .
- Зберігати потоки записів надійним способом, що відрізняється відмовою.
- Обробляти потоки записів у міру їх виникнення.

4.4 Засоби програмної реалізації рівня клієнтського застосунку

Для клієнтського рівня було використано такий набір технологій: мова програмування Dart та фреймворк Flutter.

Dart - це мова програмування, що оптимізована для розробки клієнтських застосунків на різних платформах. Він розроблений Google і використовується для створення мобільних, настільних, серверних та веб-додатків.

Flutter - фреймворк для розробки мобільних застосунків від Google. Дозволяє розробляти застосунки для Android і IOS

На рівні клієнтського застосунку було обрано технології, що дозволяють створювати застосунку для платформ Android і IOS одночасно, взаємодіяти з розробленим серверним рівнем і надавати зручний користувацький інтерфейс.

5. ОПИС ПРОГРАМНО-АПАРАТНОЇ РЕАЛІЗАЦІЇ

5.1 Опис функціоналу і формат взаємодії з агентами системи

5.1.1 Особливості розробленого агенту контролю та моніторингу поточної напруги і споживаного струму

Рис. 5.1 Структурна схема агенту контролю та моніторингу поточної напруги і споживаного струму

На рис. 5.1 можна побачити структурну схему будови даного агенту. Апаратною основою даного агенту був вибраний мікроконтролер CC1310, а також датчик ACS712.

Агент контролю та моніторингу поточної напруги і споживаного струму має такий функціонал і можливості:

- Вимірювання поточної напруги в мережі
- Вимірювання поточної сили струму, що зараз споживається
- Вимірювання спожитої електроенергії
- Реакція на досягнення граничних значень напруги в мережі
- Реакція на досягнення граничних значень споживаної сили струму

Вимірювання поточної напруги в мережі виконується за допомогою зчитування поточної напруги на піні мікроконтролера, що є пропорційною до напруги мережі за допомогою аналогового-цифрового перетворювача.

Вимірювання поточної сили струму виконується за допомогою зчитування значення аналогового датчика ACS712, що видає на пін контролера напругу пропорційну до поточного значення сили струму в мережі, за допомогою аналогового-цифрового перетворювача.

Вимірювання спожитої електроенергії виконується за рахунок математичних перетворень виміряних значень поточної напруги та поточної сили струму.

Реакцією на перевищення певних граничних значень напруги та/або сили струму може бути як відключення пристрою від мережі за допомогою реле, яке управляється за допомогою цифрового сигналу або сповіщення, що буде відправлене по мультиагентній системі.

Користувач, використовуючи користувацький додаток може змінювати такі параметри:

- Поточний стан (вкл/викл)
- Нижнє граничне значення поточної напруги в мережі
- Верхнє граничне значення поточної напруги в мережі
- Верхнє граничне значення споживаної сили струму
- Реакція системи на досягнення нижнього граничного значення поточної напруги в мережі
- Реакція системи на досягнення верхнього граничного значення поточної напруги в мережі
- Реакція системи на досягнення верхнього граничного значення споживаної сили струму

На рис 3.1 зображена діаграма прецедентів взаємодії з даним агентом

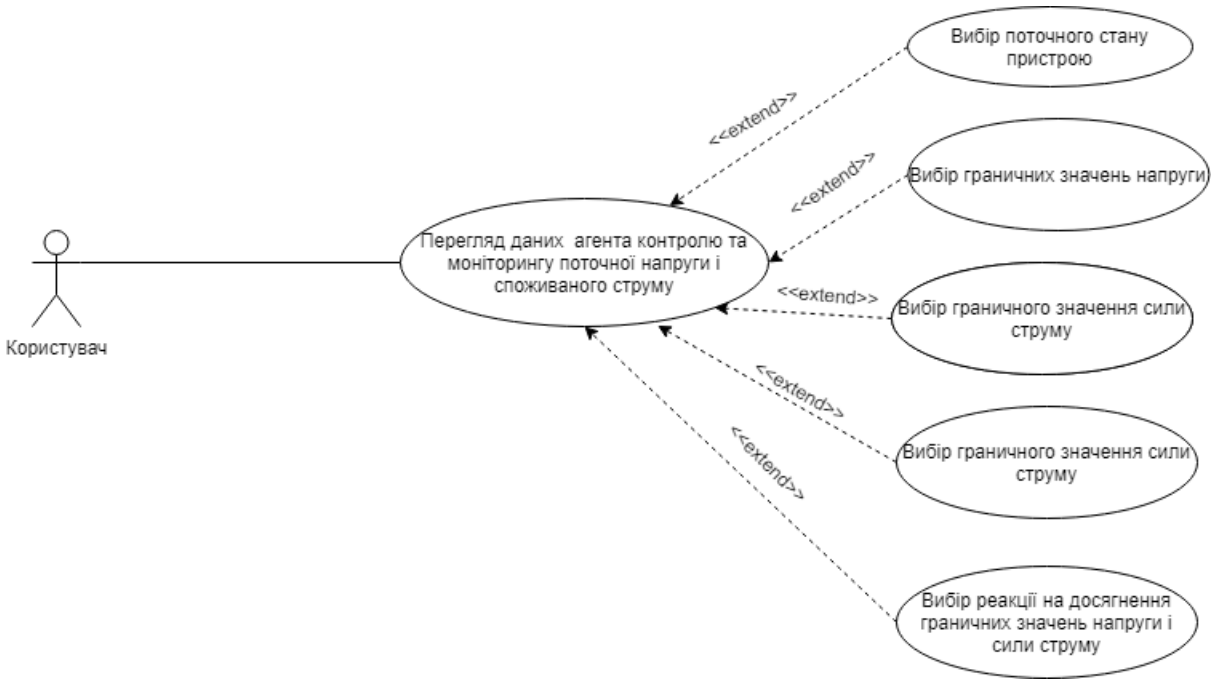

Рис 5.2 Діаграма прецедентів взаємодії користувача з агентом контролю та моніторингу поточної напруги і споживаного струму

5.1.2 Особливості розробленого агента контролю за освітленням

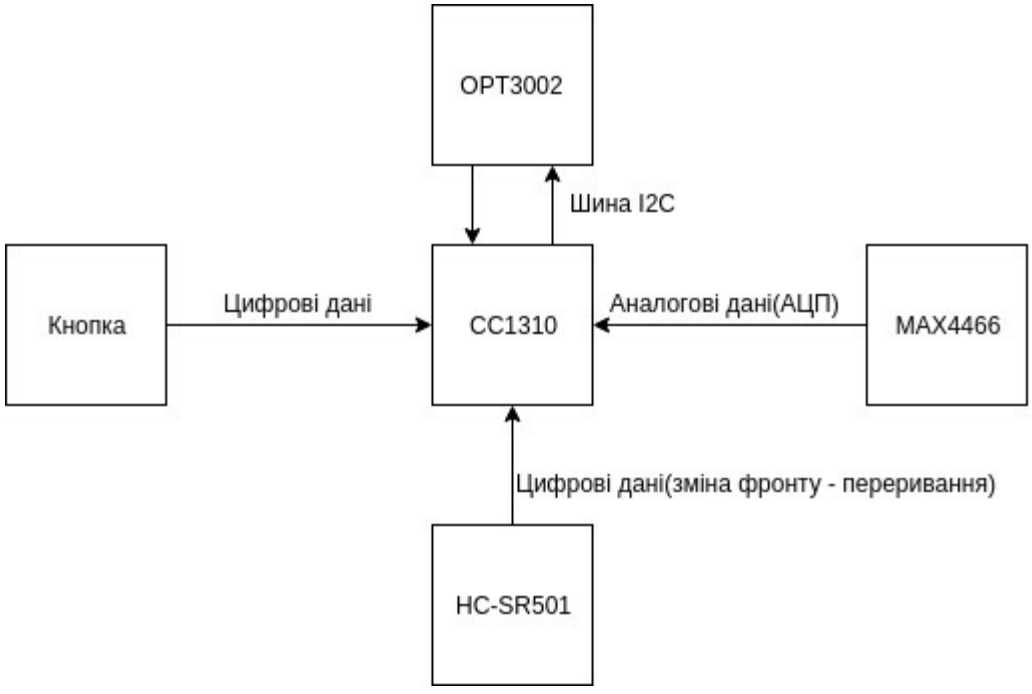

Рис. 5.3 Структурна схема агента контролю за освітленням

Агент контролю за освітленням має такий функціонал і можливості:

- Вимірювання поточної освітленості
- Керування потужністю підключених джерел світла
    - 12V
- Підтримання певного рівня освітленості
- Зміна активності приладу (включений/виключений) такими способами:
    - по кнопці
    - в певний час
    - по хлопку (голосному звуці)
    - по присутності (по руху в даному приміщенні
- Робота по графіку\

Вимірювання поточної освітленості виконується за допомогою зчитування даних з оптичного датчика OPT3002 за допомогою спілкування з ним за допомогою шини I2C.

Підтримання певного рівня освітленості виконується за рахунок керування потужністю джерел світла за допомогою зміни рівня напруги на контролюючому піні за допомогою широтно-імпульсній модуляції сигналу

Визначення моменту початку руху відбувається за рахунок реакції на зміну цифрового рівня на піні, що генерує датчик руху HC-SR501

Визначення натиску на кнопку визначається за рахунок стійкої зміни цифрового рівня сигналу на піні, до якого підключена кнопка.

Визначення хлопка(голосного звуку) відбувається за рахунок зчитування рівня поточної напруги на піні, що генерується за допомогою аналогового датчика звуку MAX4466, за допомогою аналогово-цифрового перетворювача

Робота по графіку забезпечується за рахунок контролю часу за допомогою циклічного використання таймеру з постійним періодом та первинного задання часу під час підключення до мережі мультиагентної системи.

Користувач, використовуючи користувацький додаток може змінювати такі параметри:

- Бажаний рівень освітленості
- Час ввімкнення
- Час вимикання
- Графік бажаної освітленості
- Стан (включений/виключений)
- Режим роботи (по часу включення/виключення, по графіку освітленості)

На рис 5.4 зображена діаграма прецедентів взаємодії з даним агентом

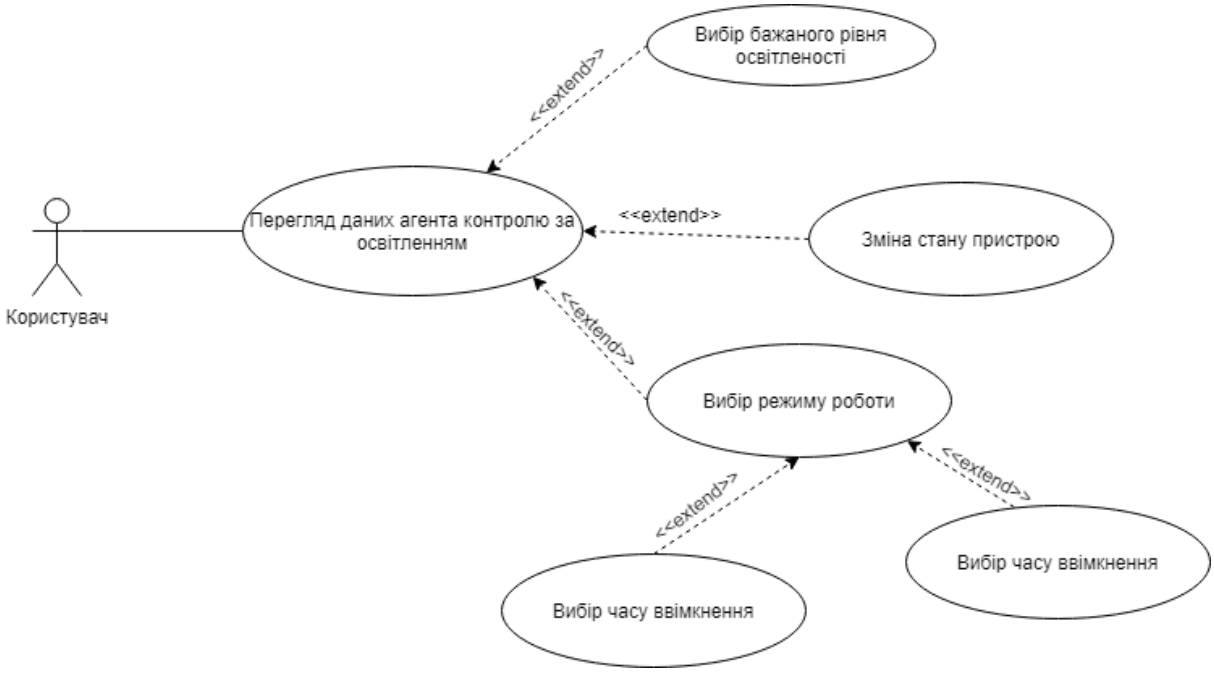

Рис 5.4 Діаграма прецедентів взаємодії користувача з агентом контролю за освітленням

5.1.3 Особливості розробленого агента контролю наявності руху в просторі

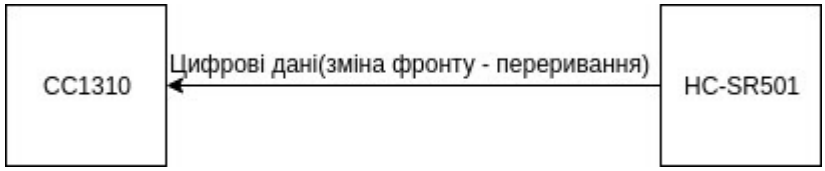

Рис. 5.5 Структурна схема агента контролю наявності руху в просторі

Розроблений датчик руху - агент контролю наявності руху в просторі призначений для використання у ролі пристрою безпеки, який визначає наявність рухомих живих істот в області виявлення, що типово визначається місцем встановлення сенсора.

Агент контролю наявності руху в просторі має такий функціонал і можливості:

- Захоплення моменту початку руху в певній області
- Захоплення продовжуваності руху в певній області
- Реакція на рух
- Реакція на низький заряд батареї
- Робота від батарейки великий проміжок часу (до 3 років)

Захоплення моменту початку руху в певній області відбувається за рахунок визначення зміни цифрового сигналу на піні, що генерується датчиком руху HC-SR501 у разі зміни параметрів, що ідентифікуються ним як рух.

Захоплення продовжуваності руху в певній області відбувається за рахунок початку відлікового таймеру під час захоплення моменту початку руху і відключення його після моменту зворотної зміни сигналу, що сповіщає закінчення руху в контрольованій області.

Реакцією на рух переважно є відправлення сповіщення про це до мережі мультиагентної системи.

Робота від батарейки протягом довгого періоду забезпечується за допомогою застосування переходу основного контроллера в режим глибокого сну і його пробудження при зміні вимірюваних параметрів.

Користувач, використовуючи користувацький додаток може змінювати такі параметри:

- Реакція на сповіщення про рух
- Умови сповіщення:
    - початок руху
    - рух певної продовжуваності
- Графік роботи
- Реакція на низький рівень батареї

На рис 5.6 зображена діаграма прецедентів взаємодії з даним агентом

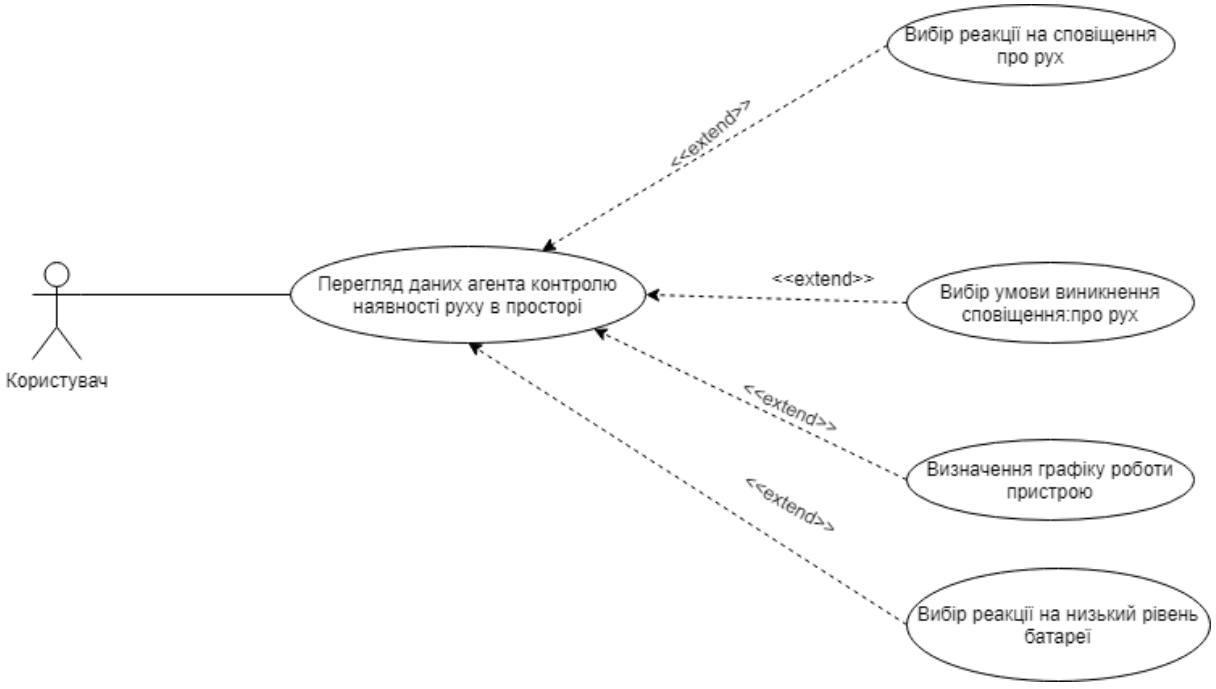

Рис 5.7 Діаграма прецедентів взаємодії користувача з агентомонтролю наявності руху в просторі

5.1.4 Особливості розробленого агента збору інформації про кількість відвідувань приміщення

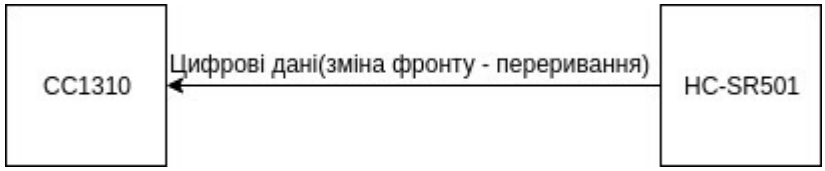

Рис. 5.8 Структурна схема агента збору інформації про кількість відвідувань приміщення

Агент збору інформації про кількість відвідувань приміщення має такий функціонал і можливості:

- Фіксація моменту відкриття дверей
- Фіксація моменту закриття дверей
- Реакція на відкриття дверей
- Реакція на закриття дверей
- Реакція на низький заряд батареї
- Робота від батарейки великий проміжок часу (до 3 років)

Фіксація моменту відкриття дверей відбувається шляхом виявлення зміни цифрового сигналу на піні, шляхом обробки переривання по зміні фронту сигналу на піні.

Фіксація моменту закриття дверей відбувається шляхом виявлення зворотної зміни цифрового сигналу на піні , шляхом обробки переривання по зміні фронту сигналу на піні.

Робота від батарейки протягом довгого періоду забезпечується за допомогою застосування переходу основного контроллера в режим глибокого сну і його пробудження при зміні вимірюваних параметрів.

Користувач, використовуючи користувацький додаток може змінювати такі параметри:

- Реакція на відкриття
- Реакція на закриття
- Тип реакції на низький рівень батареї

На рис 5.9 зображена діаграма прецедентів взаємодії з даним агентом

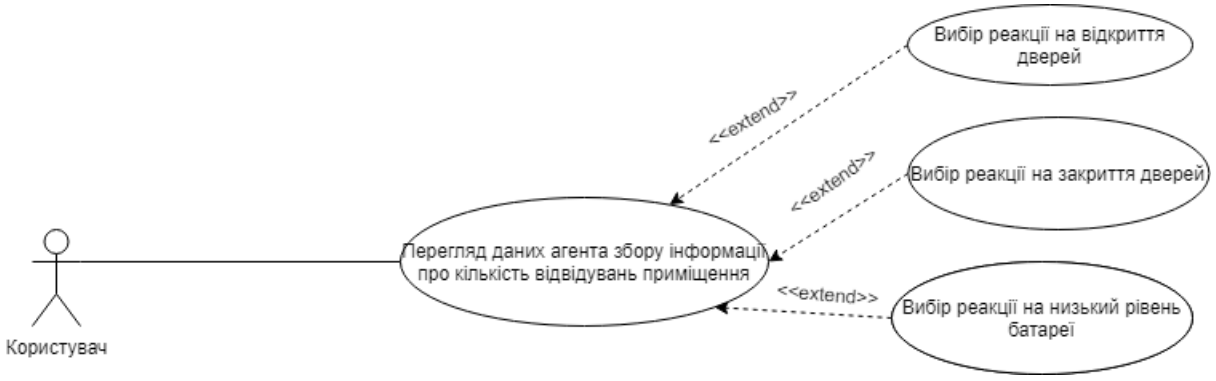

Рис 5.9 Діаграма прецедентів взаємодії користувача з агентом збору інформації про кількість відвідувань приміщення

5.1.5 Особливості агента збору даних про мікроклімат

Рис. 5.10 Структурна схема агента збору даних про мікроклімат

Агент збору даних про мікроклімат має такий функціонал і можливості:

- Збір даних температури
- Збір даних вологості
- Реакція на досягнення граничних значень температури
- Реакція на досягнення граничних значень вологості
- Реакція на низький заряд батареї
- Робота від батарейки великий проміжок часу (до 3 років)

Вимірювання поточної температури виконується за допомогою зчитування даних з датчика HDC1080 за допомогою спілкування з ним за допомогою шини I2C.

Вимірювання поточної вологості виконується за допомогою зчитування даних з датчика HDC1080 за допомогою спілкування з ним за допомогою шини I2C.

Визначення моменту початку руху відбувається за рахунок реакції на зміну цифрового рівня на піні, що генерує датчик руху HC-SR501

Визначення натиску на кнопку визначається за рахунок стійкої зміни цифрового рівня сигналу на піні, до якого підключена кнопка

При досягненні певних граничних значень вологості і температури виникає сповіщення про це, що відправляється до мережі мультиагентної системи.

Користувач, використовуючи користувацький додаток може змінювати такі параметри:

- Частота вимірювання температури
- Частота вимірювання вологості
- Нижнє граничне значення температури
- Верхнє граничне значення температури
- Нижнє граничне значення вологості
- Верхнє граничне значення вологості
- Тип реакції при перетині
    - нижньої границі температури
    - нижньої границі вологості
    - верхньої границі температури
    - верхньої границі вологості
- Тип реакції на низький рівень батареї

На рис 5.11 зображена діаграма прецедентів взаємодії з даним агентом

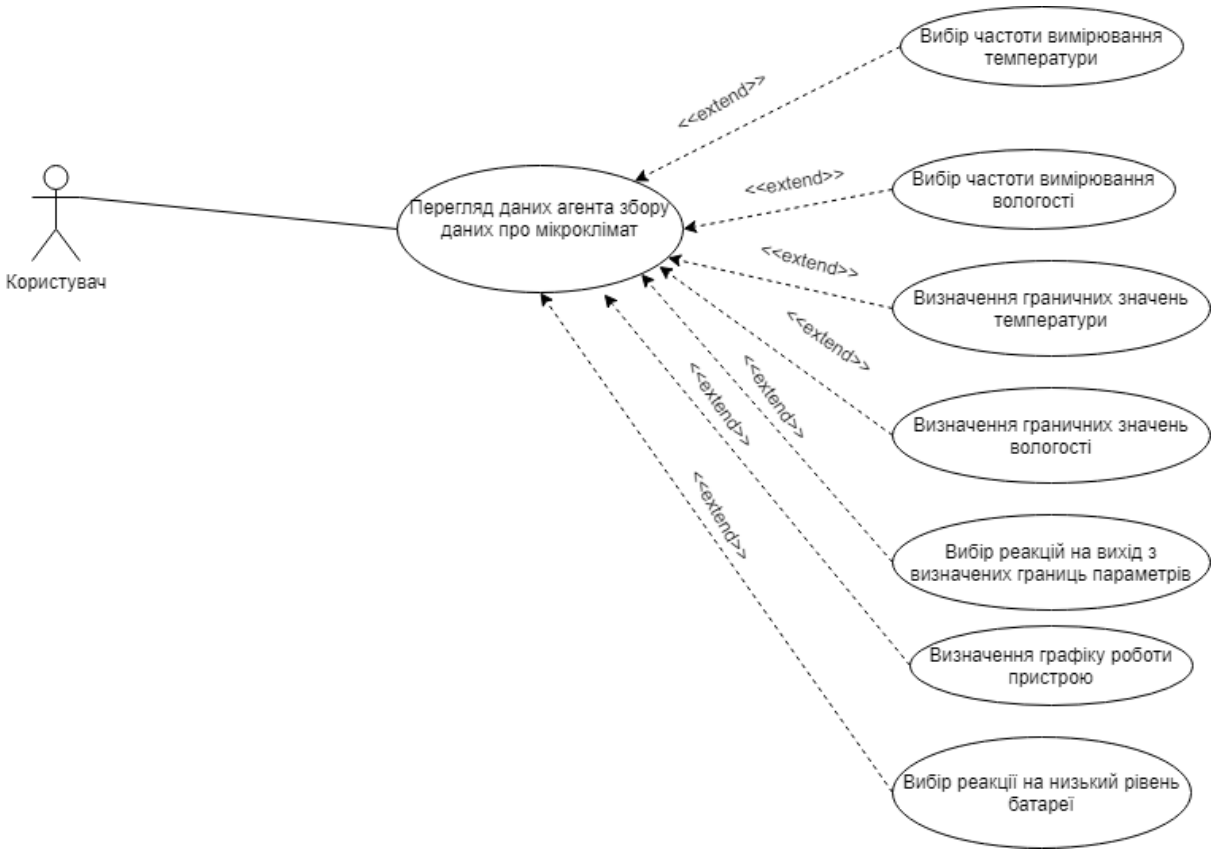

Рис 5.11 Діаграма прецедентів взаємодії користувача з агентом збору даних про мікроклімат

5.1.6 **Особливості розробленого агенту контролю цілісності системи водопостачання**

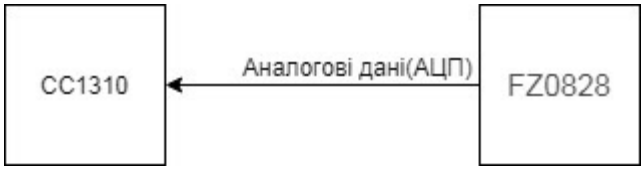

Рис. 5.12 Структурна схема агента контролю цілісності системи водопостачання

Агент контролю цілісності системи водопостачання має такий функціонал і можливості:

- Виявлення появи вологи на провідній області
- Закриття клапану при протіканні
- Реакція на протікання
- Реакція на низький заряд батареї

- Робота від батарейки великий проміжок часу (до 3 років)

Виявлення появи вологи на провідній області відбувається за рахунок зчитування поточної напруги на піні, до якого підключений FZ0828, за допомогою аналогово-цифрового перетворювача, і порівняння даного значення з попередньо визначеним граничним значенням, що характеризує певний рівень вологи на цій області.

При виявленні протікання за допомогою цифрового сигналу перекривається клапан, що підключений до певного управляючого піна.

Користувач, використовуючи користувацький додаток може змінювати такі параметри:

- Тип реакції на протікання
- Тип реакції на низький рівень батареї

На рис 5.13 зображена діаграма прецедентів взаємодії з даним агентом

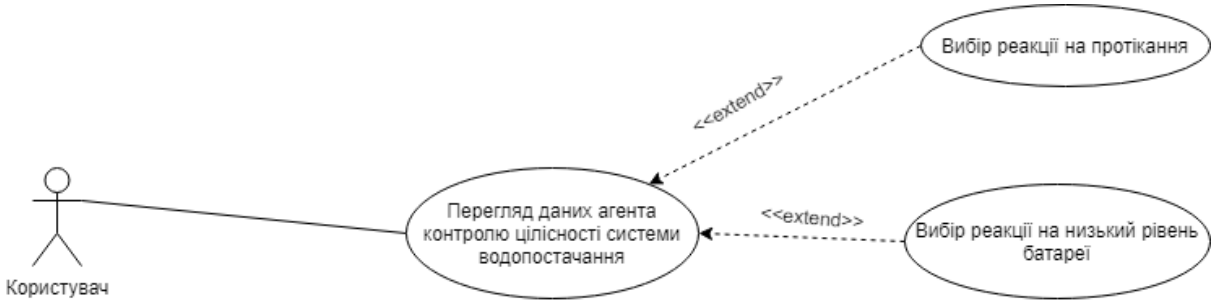

Рис 5.13 Діаграма прецедентів взаємодії користувача з агентом контролю цілісності системи водопостачання

5.2 Опис обміну інформацією всередині системи

Для взаємодії між агентами та програмними модулями системи використовуються повідомлення, що сформовані за протоколом Google Protocol Buffers та зашифровані за допомогою алгоритму AES-128.

Кожен агент даної мультиагентної системи отримує і відсилає повідомлення певних типів.

Для кожного з типів повідомлень агент має метод і алгоритм його обробки.

5.3 Опис структури і функціоналу серверного рівня системи

Серверний рівень мультиагентної системи виступає у ролі середньої ланки, яка оброблює і зберігає дані, що надходять із двох різних рівнів системи, а саме з рівня мультиагентної системи та рівня мобільного застосунку. Серверний рівень мультиагентної системи побудований на мікросервісній архітектурі для забезпечення масштабування системи з додаванням нових типів пристроїв, а також забезпечення більшої стійкості системи. Задля зв'язку мобільного додатку з сервером для кожного з типів пристроїв існує сервіс API, Кожен з таких сервісів має свою базу даних, що зберігає дані про даний тип пристроїв, а також API, яке по певним запитам надає інформацію про список доступних пристроїв користувачу, про поточну конфігурацію заданого пристрою, про події, що згенерував пристрій, про поточний графік роботи пристрою та інше. Особливими сервісами на рівні API є сервіс, мета якого зберігати і забезпечувати отримання даних користувача, а також надавати можливість реєстрації і авторизації користувачів, сервіс, мета якого зберігати і забезпечувати отримання даних кімнат і сервіс, мета якого зберігати і забезпечувати отримання даних пристроїв - хабів. Також друга група сервісів - це сервіси, що забезпечують проміжну ланку між API і MQTT брокером, що доставляє повідомлення вже до межового роутера системи. Це сервіси, які трансформують запит надісланий користувачем з мобільного застосунку до кінцевого пристрою,у формат, що є правильним для розуміння межовим роутером. Третя група сервісів - це сервіси, що забезпечують трансформацію повідомлень надісланих з межового роутера системи до формату, що буде зрозумілий для сервісів API, які зможуть обробити і зберегти інформацію, надіслану з кінцевого пристрою. Окремим сервісом

є сервіс MQTT брокера, що забезпечує надійний зв’язок з межовими роутерами мультиагентних систем. Також існує ще два технічних мікросервіси, що забезпечують роботу всієї мікросервісної системи, а саме Zuul Proxy Server і Eureka Server. На рис. 5.14 зображена дана структура сервісів, що забезпечують роботу даного серверного рівня даної мультиагентної системи.

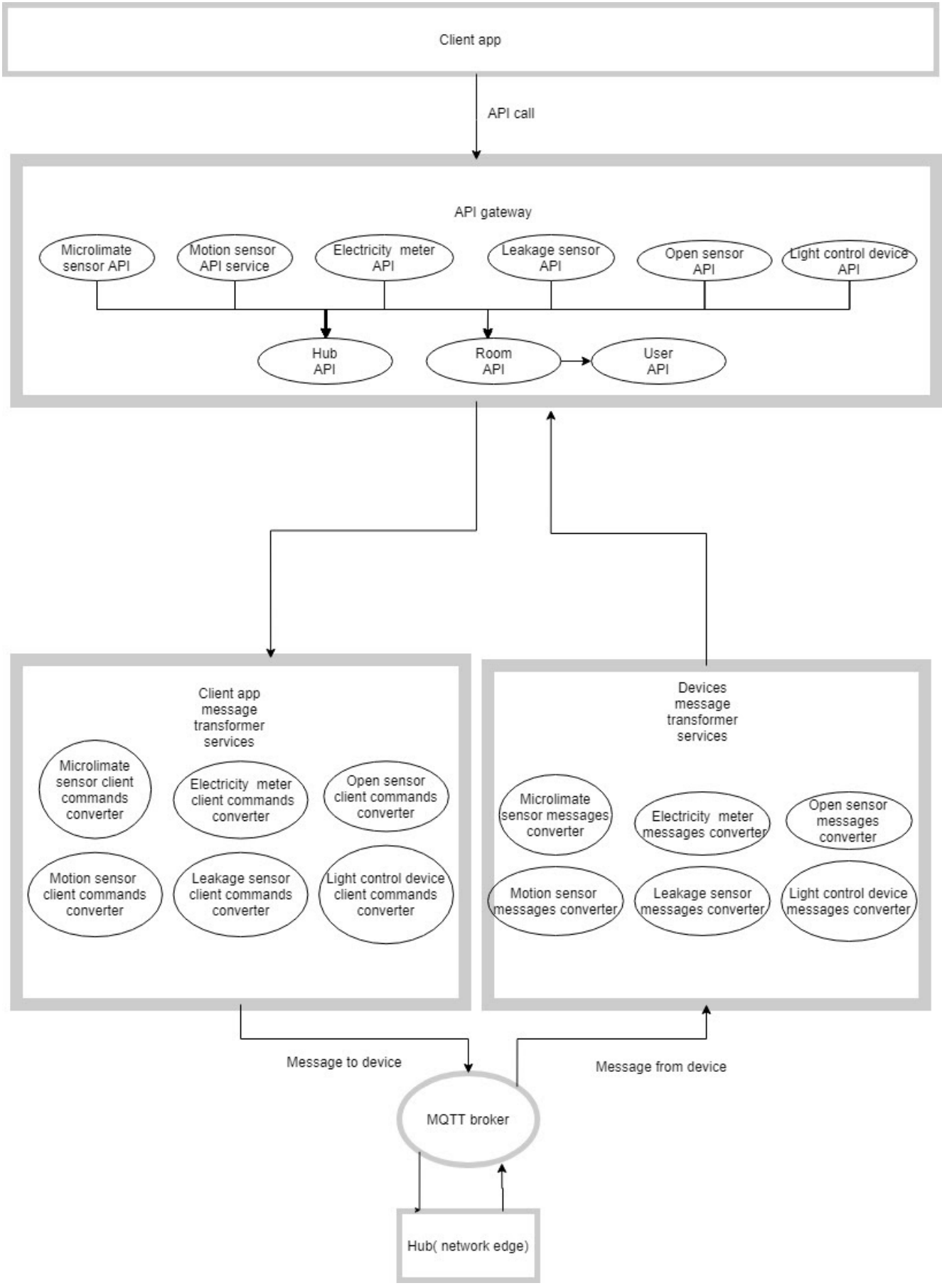

Рис. 5.14 Структурна схема сервісів серверного рівня

Для забезпечення легкого масштабування системи було спроектовано і реалізована ієрархія класів, що пов’язує основні характеристики і параметри розумних пристроїв, що дозволяє для створення пристрою нового типу, продовжити розроблену базову ієрархію.На рис. 5.15 зображена базова ієрархія сутностей,що повністю описують параметри розумних пристроїв.

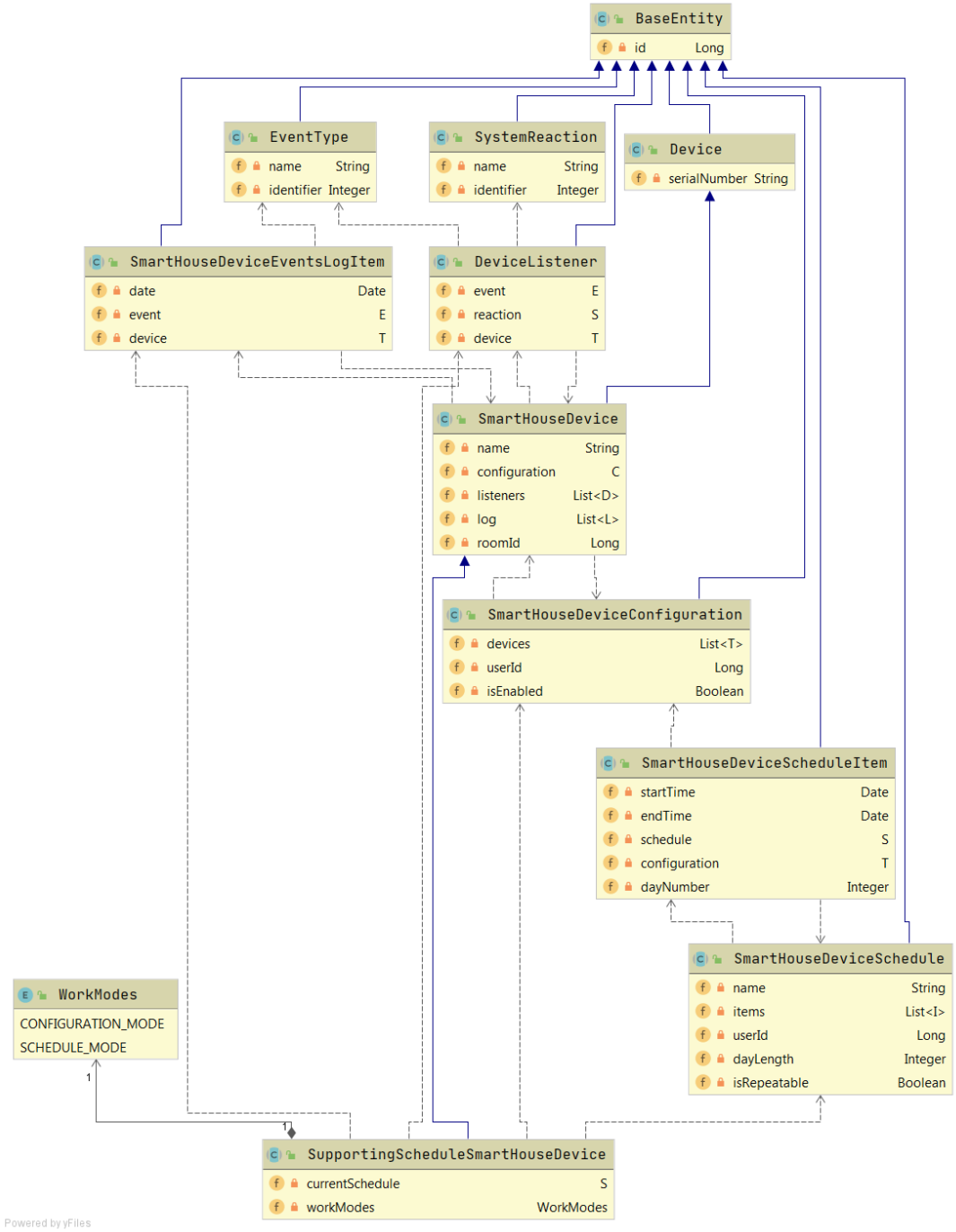

Рис. 5.15 Базова ієрархія сутностей агента мультиагентної системи

На рис. 5.15 зображено приклад розширення базової ієрархії для забезпечення ієрархії сутностей для агента контролю за відкриттям дверей

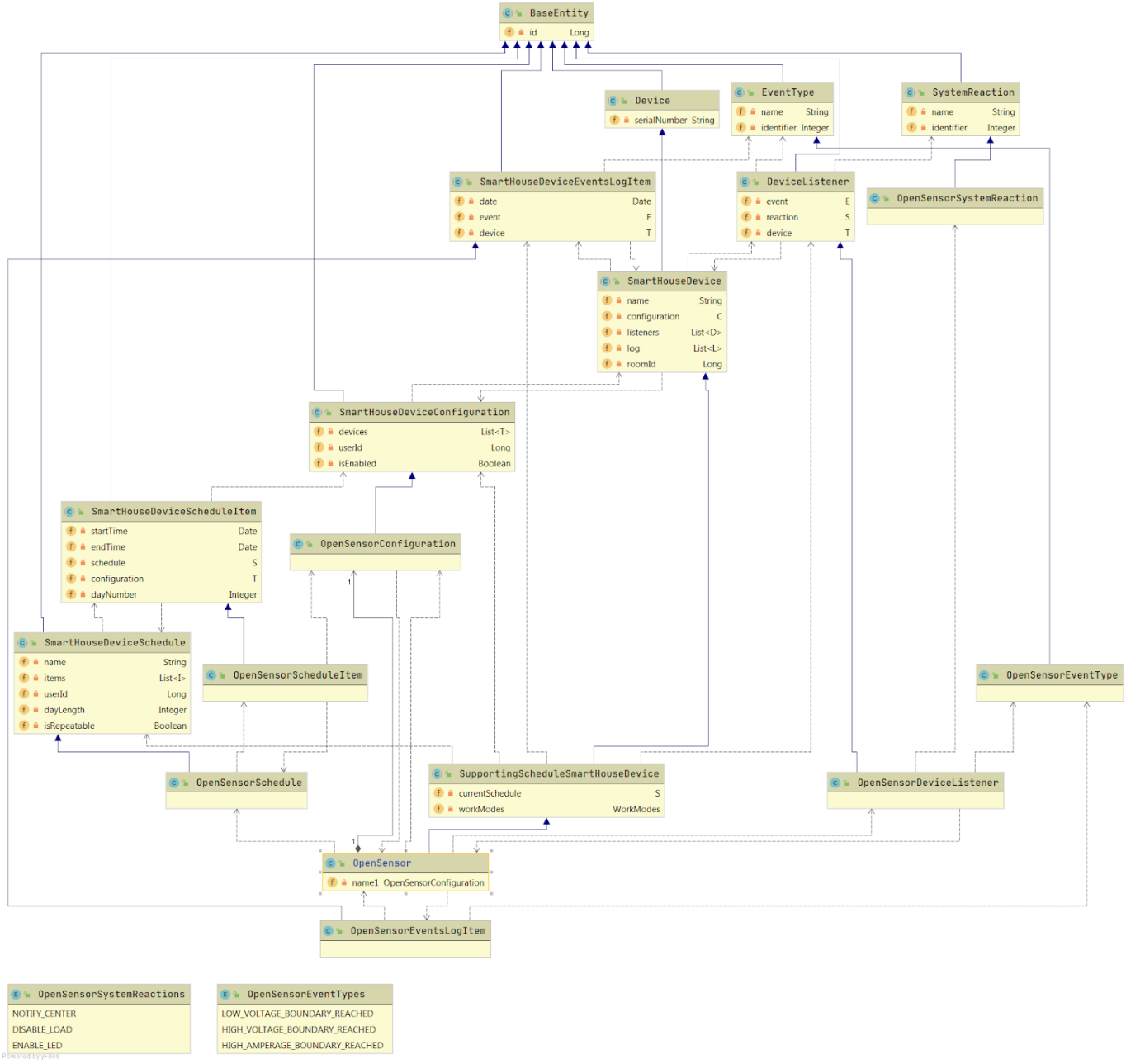

Рис. 5.15 Ієрархія сутностей агента відкриття дверей

Для забезпечення масштабування сервісів API для агентів системи була розроблена ієрархія базових контролерів REST, що забезпечують базові методи для отримання даних про агент і вона зображена на рис. 5.16

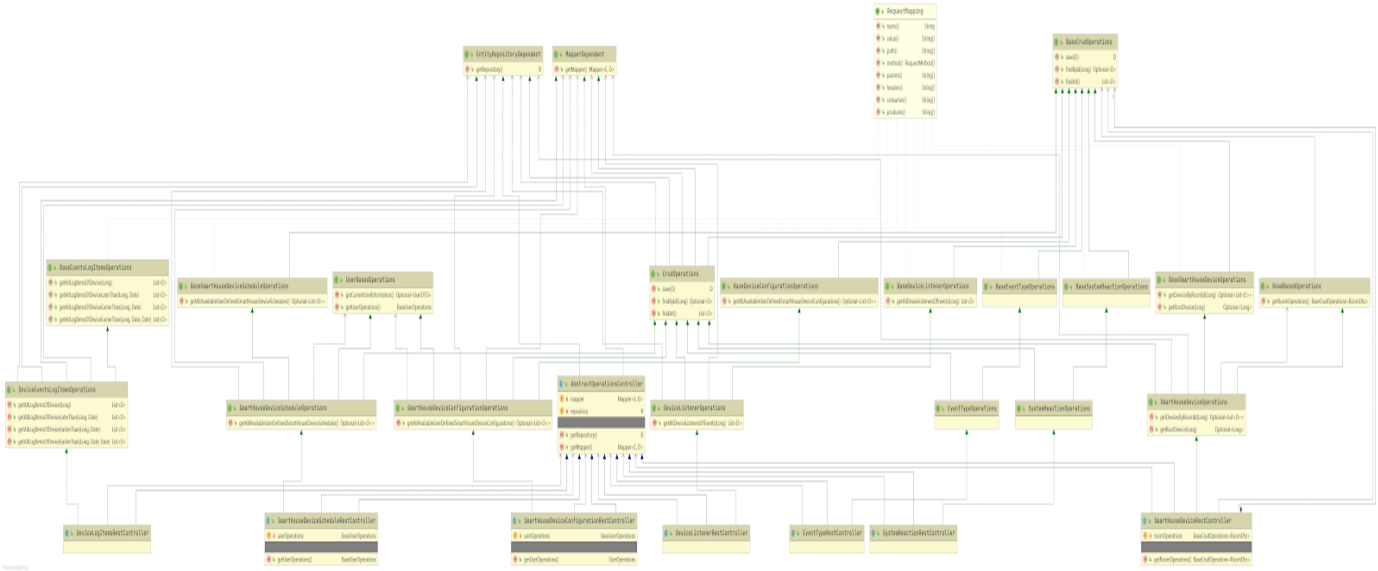

Рис. 5. 16 Базова ієрархія REST-контролерів для агента мультиагентної системи

5.4 Опис функціоналу мобільного додатку

Мобільний застосунок - це користувацький клієнт мультиагентної системи, що надає користувачу доступ до такого функціоналу:

- авторизація користувача
- реєстрація користувача
- перегляд підключених будинків
- додавання нових будинків
- перегляд наявних пристроїв в будинку
- перегляд даних датчику руху
- зміна параметрів роботи датчика руху
- перегляд даних агента збору даних про мікроклімат
- зміна параметрів роботи агента збору даних про мікроклімат
- перегляд даних агента контролю цілісності системи водопостачання
- зміна параметрів роботи агента контролю цілісності системи водопостачання
- перегляд даних датчика відкриття
- зміна параметрів роботи датчика відкриття

- перегляд даних агента контролю за освітленням
- зміна параметрів роботи агента контролю за освітленням
- перегляд даних агента контролю за енергоспоживанням
- зміна параметрів роботи агента контролю за енергоспоживанням

Програмний застосунок для керування мультиагентною системою містить у собі одного головного актора – користувач системи;

На рисунку 5.17 представлена діаграма прецедентів, яка описує загальні функції та дії актора у системі.

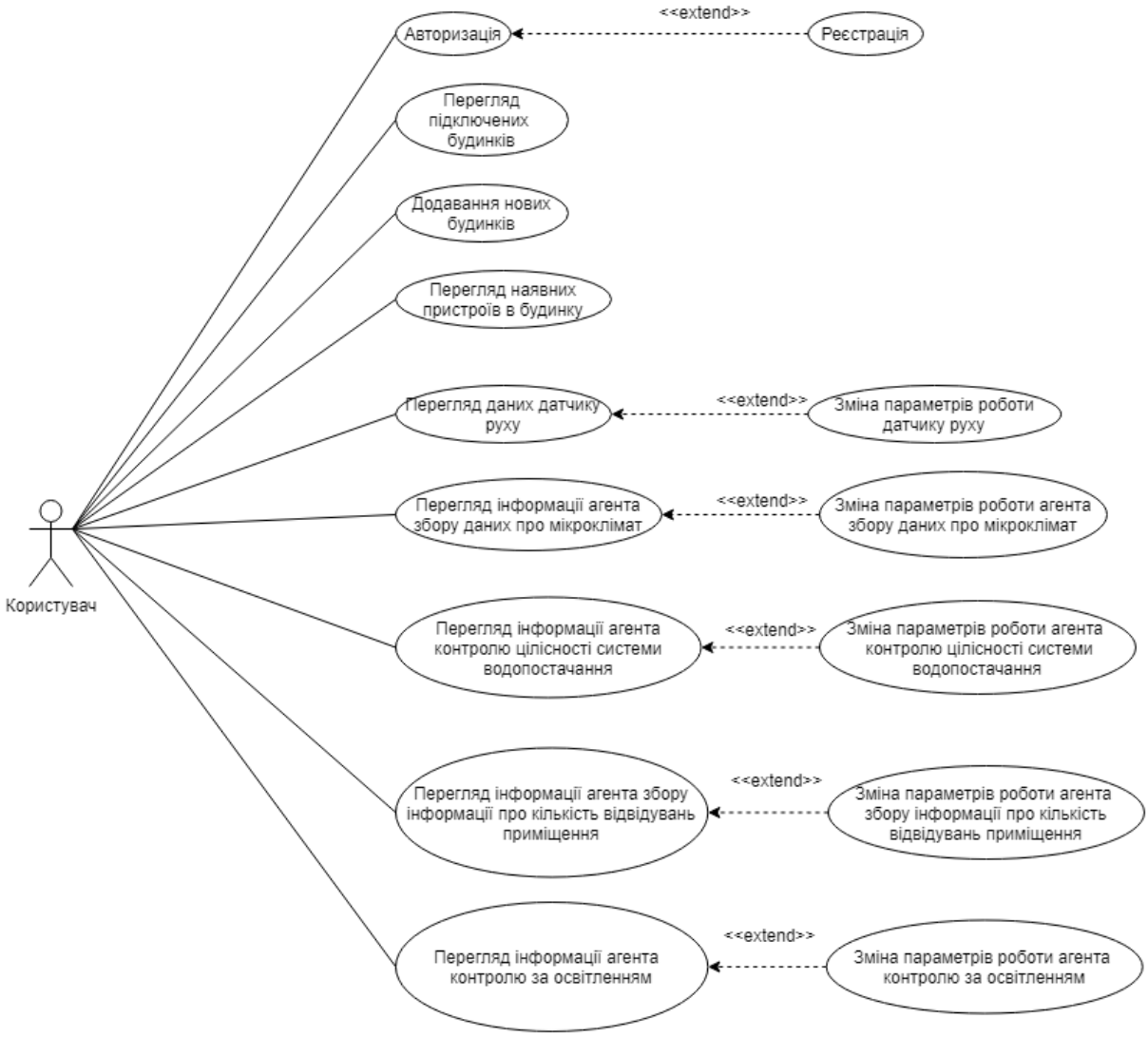

Рисунок 5.17 — Загальна діаграма прецедентів

6. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблена програмний комплекс інтерфейсу розроблений для мобільних платформ і призначений для використання на пристроях IOS та Android.

6.1 Інсталяція та системні вимоги

Для встановлення додатку його необхідно завантажити на пристрій та встановити методами операційної системи. Також необхідно мати стабільний доступ до інтернету з швидкістю не менше ніж 5 мбіт за секунду для стабільної роботи передачі даних. Для використання можливостей програми віддалено керувати платформою необхідно надати їй доступ до роботи з мережею.

6.2 Інструкція з використання програмного продукту

При вході до мобільного застосунку , користувачеві необхідно авторизуватися, Для авторизації користувачу потрібно ввести свій логін і пароль. Логіном користовача може бути як і електронна адреса реєстрації, так і вибране ім’я при реєстрації. У випадку, якщо введений пароль не підходить, у користувача є можливість відновити пароль.

Якщо користувач не зареєстрований, в системі потрібно пройти етап реєстрації, заповнивши стандартну форму, у якій вказати свою електронну адресу, бажаний пароль, ім’я та бажаний нік.

Після успішного входу користувача до системи користувачу показується екран його поточних контрольованих систем. Для кожної системи відображається її ім’я і коротка поточна інформація про поточний стан. На цьому ж екрані можна додати нову контрольовану будівлю, натиснувши кнопку “Додати”. Після цього на екрані відображається зчитувач QR-коду, яким користувач має зчитати код, що розташований на задній панелі пристрою хабу. Після успішного зчитування QR-коду, користувачу буде запропоновано ввести назву цієї будівлі і натиснути кнопку для закінчення реєстрації системи.

Після вибору користувачем будь-якої з контрольованих систем користувач потрапляє на наступну сторінку, на якій відображається список відділень будівлі (кімнат), якщо вони були створені, або поточний стан всієї будівлі, а саме яка температура, вологість, освітленість чи були останнім часом аварії, чи були певні сповіщення від системи, а також стан певних спеціалізованих пристроїв контролю і моніторингу такі як розумні розетки та інші. Також на даній сторінці користувач має можливість створити окреме відділення (кімнату) в контрольованій будівлі, де має ввести лише ім’я з яким вона буде відображатися на основній сторінці. Після цього користувач має можливість може натиснути на відділення і потрапити на сторінку, де побачить більш детальну інформацію саме про дане відділення аналогічне до інформацію про всю будівлі.

Також користувач має можливість додати новий пристрій контролю та моніторингу натиснувши на кнопку “Додати пристрій”, після чого користувачу відображається зчитувач QR-коду, яким аналогічно до реєстрації будівлі потрібно зчитати QR-код розташований на задній панелі пристрою. Після цього користувачу відобраєається форма, в якій потрібно ввести назву пристрою(необов’язково) та вибрати відділення до якого користувач бажає додати пристрій (необов’язково). Після даного кроку пристрій буде приєднаний до мультиагентної мережі.

Користувач, який знаходиться на сторінці даних про відділення або про систему в цілому, може змінювати параметри контролю різних підключених систем.

Наприклад, натиснувши на блок поточної температури і вологості, користувач має можливість вибрати граничні значення температури та вологості та при бажанні спеціалізовано налаштувати реакцію системи на перетин даних параметрів. Також для кожного параметру можна задати графік його значення, а такоє скласти дану послідовність на різний період(від 1 дня). Також на даній сторінці користувач може побачити намальований графік зміни параметрів, середнє значення за певний період.

Натиснувши на дані про поточну освітленість, користувач може вибрати бажану освітленість, за допомогою слайдера, а також при бажанні вибрати графік освітленості в даному відділенні. Також можна вибрати особливі реакції системи на певні зміни в даному параметрі.

При виборі у меню певної розумної розетки, користувач має можливість змінити її поточний стан(включити/виключити), а також налаштувати максимальну потужність, шо можна підключати до даної розетки, а також не даному екрані можна вибрати специфічні реакції на перевищення визначених параметрів. Також на даному екрані можна побачити поточну споживану потужність, а також споживання електроенергії пристроями, підключеними до даної розетки протягом різного періоду часу.

Для контролю датчиків руху потрібно натиснути на відповідний блок на загальній сторінці відділення,на якій можна побачити список подій, що відбувалися, та їх час, а також налаштувати графік необхідного виявлення руху, а також рівень важливості виявлення в певний час. Якщо користувач для певного часу вибере високу важливість виявлення, то при виявленні руху системою, сповіщення про це буде відправлено на телефон у вигляді push-повідомлення, а також виконається певна специфічна дія, яка може бути вибрана користувачем.

Для контролю датчиків відкриття потрібно натиснути на відповідний блок на загальній сторінці, що дозволить побачити певну статистику, а також аналогічно до контролю руху, налаштувати певний графік контролю за даними параметрами

відкриття і налаштувати спеціалізовані реакції на певні процеси з контрольованими параметрами.

Для контролю датчиків протікання потрібно натиснути на відповідний блок на загальній сторінці, що дозволить побачити певну статистику протіквнь, а також аналогічно до контролю відкриття, налаштувати певний графік контролю за даними параметрами відкриття і налаштувати спеціалізовані реакції на певні процеси.

Також користувач має можливість налаштувати інших користувачів, що можуть мати до контролю і моніторингу ситуацією над будівлею або лише над певною її частиною(відділенням). Для цього на загальній сторінці будівлі чи сторінці кімнати потрібно натиснути кнопку “Додати користувача” після чого буде відображена форма, на якій потрібно ввести нік або електронну адресу користувача, у висхідному меню вибрати знайдений варіант і потім вибрати права даного користувача в даному відділенні будівлі

ВИСНОВКИ

1. У ході аналізу необхідних характеристик, що визначають принципи роботи мультиагентної системи, були визначені чіткі параметри, що характеризують пристрої мультиагентної системи. Аналіз показав, що дані параметри є спільними для всіх пристроїв.

2. Було проведено моделювання архітектури різних рівнів системи, а саме рівня кожного агента, мережевого рівня, рівня серверного застосунку, рівня мобільного додатку. Були визначені характеристики, що роблять систему більш надійною, масштабованою і гнучкою

3. Було досліджено і проаналізовано різні засоби програмно-апаратної реалізації розробленої архітектури. Було обрано і обґрунтовано вибір найбільш оптимальних і гнучких засобів, що забезпечують виконання вимог до розроблюваного програмно-апаратного комплексу

4. Розроблений програмно-апаратний комплекс відповідає визначеним вимогам та дозволяє використовувати та доповнювати інтелектуальну мультиагентну систему для задач, що існують і виникають для потреб контролю над параметрами навколишнього середовища

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.TIDA027–September 2018 .6LoWPAN Mesh Designs Texas Instruments Application report

2. CC1310 Reference manual Texas Instruments.

3. .Texas Instruments TI-RTOS Reference Guide

4. https://docs.spring.io/ Spring Framework Reference Documentation

5. https://spring.io/projects/spring-cloud Spring Cloud Reference Documentation.

6. https://kafka.apache.org/ Apacha Kafka Message Broker Reference Manual

7. CC13x0 SimpleLink™ TI 15.4-Stack 2.x.x Embedded Developer's Guide

8. 6LoWPAN demystified, Jonas Olsson, 2014

ДОДАТОК 1

Розробка мультиагентної бездротової Mesh – мережі Smart пристроїв

Специфікація

НТУУ «КПІ» ТЕФ АПЕПС ТІ-61

Листів 1

Київ - 2020

Позначення	Найменування	Примітки
Документація		
Записка	Записка.docx	Текстова частина дипломної роботи
Компоненти		
Програмний код		Текст програмного продукту
Опис		Опис програмного продукту

ДОДАТОК 2

Розробка мультиагентної бездротової Mesh – мережі Smart пристроїв

Лістинг програмного модуля

НТУУ «КПІ» ТЕФ АПЕПС ТІ-61

Листів 17

Київ - 2020

//schedule.c

```
#include "schedule.h"
static void Schedule_findAndSetCurrentScheduleItem(ScheduleHandle schedule);
void Schedule_timeChanged(Time time, void *arg)
{
   ScheduleHandle schedule = (ScheduleHandle) arg;
   Time *currentTime = &schedule->calendar->time;

   if (Time_isMidnight(currentTime))
      {
      schedule->currentScheduleItemIndex = NO_SCHEDULE_ITEM;
      }
   DaySchedule *daySchedule =
         &schedule->weekSchedule.daySchedule[currentTime->weekDayNumber];
   if (daySchedule->scheduleItemsCount > 0)
   {
      uint16_t nextScheduleItemIndex = schedule->currentScheduleItemIndex + 1;
      if (nextScheduleItemIndex < daySchedule->scheduleItemsCount)
      {
         ScheduleItemBase *nextScheduleItemBase =
               schedule->retrieveScheduleItemBaseFunction(
                     daySchedule->items,
                     schedule->currentScheduleItemIndex + 1);
         if (compareTime(currentTime, &nextScheduleItemBase->startTime) == 0)
         {
            schedule->currentScheduleItemIndex = nextScheduleItemIndex;
            if (schedule->scheduleItemChangeCallback != NULL)
            {
               schedule->scheduleItemChangeCallback(nextScheduleItemBase,
                                          schedule->callbackArg);
            }
         }
      }
   }
```

```
}
ScheduleHandle Schedule_init(Time current, ScheduleItemChangedCallback callback,
            void *callbackArg,
            RetrieveScheduleItemBaseFunction retrieveScheduleItemBaseFunction)
{
   ScheduleHandle schedule = calloc(1, sizeof(ScheduleConfig));
   schedule->calendar = Calendar_init(current, Schedule_timeChanged, schedule);
   schedule->currentScheduleItemIndex = NO_SCHEDULE_ITEM;
   schedule->scheduleItemChangeCallback = callback;
   schedule->callbackArg = callbackArg;

   if (retrieveScheduleItemBaseFunction == NULL)
         {
            return NULL;
         }
         schedule->retrieveScheduleItemBaseFunction =
               retrieveScheduleItemBaseFunction;
   return schedule;
}
void Schedule_updateCurrentTime(ScheduleHandle schedule, Time time)
{
   Calendar_stop(schedule->calendar);
   schedule->calendar->time = time;
   Calendar_start(schedule->calendar);
}
void Schedule_setScheduleItemChangeCallback(
      ScheduleHandle schedule, ScheduleItemChangedCallback callback,
      void *arg)
{
   schedule->scheduleItemChangeCallback = callback;
   schedule->callbackArg = arg;
}
void Schedule_setDaySchedule(ScheduleHandle schedule, uint8_t weekDay,
                     DaySchedule *daySchedule, ssize_t scheduleItemSize)
{
   if (weekDay < WEEK_DAYS_COUNT)
   {
```

```
        DaySchedule *currentDaySchedule =
                &schedule->weekSchedule.daySchedule[weekDay];
        ssize_t dayScheduleItemsSize = scheduleItemSize
                * daySchedule->scheduleItemsCount;
        currentDaySchedule->items = malloc(dayScheduleItemsSize);

        memcpy(currentDaySchedule->items, daySchedule->items,
              dayScheduleItemsSize);
        currentDaySchedule->scheduleItemsCount =
                daySchedule->scheduleItemsCount;

    }
    if (weekDay == schedule->calendar->time.weekDayNumber)
    {
        Schedule_findAndSetCurrentScheduleItem(schedule);
    }
}
void Schedule_setWeekSchedule(ScheduleHandle schedule,
                              WeekSchedule weekSchedule,
                              ssize_t scheduleItemSize)
{
    uint8_t i;
    for (i = 0; i < WEEK_DAYS_COUNT; i++)
    {
        Schedule_setDaySchedule(schedule, i, &weekSchedule.daySchedule[i],
                                scheduleItemSize);
    }
}
void Schedule_findAndSetCurrentScheduleItem(ScheduleHandle schedule)
{
    Time * currentTime = &schedule->calendar->time;
    DaySchedule *currentDaySchedule =
          &schedule->weekSchedule.daySchedule[currentTime->weekDayNumber];
    uint32_t currentTimeInSeconds = Time_getSeconds(currentTime);
    uint16_t start, end, m;
    if (currentDaySchedule->scheduleItemsCount > 0)
    {
```

```
    start = 0;
    end = currentDaySchedule->scheduleItemsCount;
    while (start < end - 1)
    {
        m = (start + end) / 2;
        if (currentTimeInSeconds
                < Time_getSeconds(
                        &schedule->retrieveScheduleItemBaseFunction(
                                currentDaySchedule->items, m)->startTime))

        {
            end = m;
        }
        else
        {
            start = m;
        }
    }
    schedule->currentScheduleItemIndex = start;
        ScheduleItemBase *currentBase =
                schedule->retrieveScheduleItemBaseFunction(
                        currentDaySchedule->items,
                        schedule->currentScheduleItemIndex);
        schedule->scheduleItemChangeCallback(currentBase,
                schedule->callbackArg);
    }
}
//calendar.c

#include "calendar.h"
int compareTime(Time *time1, Time *time2)
{
    uint32_t time1InSec = Time_getSeconds(time1);
    uint32_t time2InSec = Time_getSeconds(time2);
    int subtract = time1InSec - time2InSec;
    if (subtract < 0)
    {
```

```
        return -1;
}
else if (subtract == 0)
{
   return 0;
}
else
   return 1;
}
uint32_t Time_getSeconds(Time *time)
{
return time->second + time->minute * 60 + time->hour * 3600;
}
bool Time_isMidnight(Time * time)
{
return time->hour == 0 && time->minute == 0 && time->second == 0;
}
static void Calendar_timeHandler(union sigval val)
{
Calendar *calendar = (Calendar *) val.sival_ptr;
   calendar->time.second++;
   if (calendar->time.second == 60)
{
       calendar->time.second = 0;
       calendar->time.minute++;
       if (calendar->time.minute == 60)
   {
           calendar->time.minute = 0;
           calendar->time.hour++;
           if (calendar->time.hour == 24)
      {
              calendar->time.weekDayNumber++;
              if (calendar->time.weekDayNumber == 7)
          {
                  calendar->time.weekDayNumber = 0;
          }
       }
```

```
    }
}
if (calendar->callback != NULL)
{
    calendar->callback(calendar->time, calendar->callbackArg);
}
}
static void Calendar_configure(Calendar *calendar)
{

Calendar_start(calendar);
}
Calendar * Calendar_init(Time startTime, TimeChangedHandler changeHandler,
                    void *callbackArg)
{
Calendar * calendar = calloc(1, sizeof(Calendar));
calendar->time = startTime;
calendar->callback = changeHandler;
calendar->callbackArg = callbackArg;
Calendar_configure(calendar);
return calendar;

}
void Calendar_stop(Calendar * calendar)
{
Utils_cancelTimer(&calendar->calendarTimer);
}
void Calendar_start(Calendar *calendar)
{
Utils_setupTimerWithCallback(Calendar_timeHandler, calendar,
                        &calendar->calendarTimer, 1, 0, 1);
}



//MicroclimateSensor.c
#include <MicroclimateSensor.h>
static void MicroclimateSensor_turnedOff(MicroclimateSensor * device);
```

```
static void MicroclimateSensor_turnedOn(MicroclimateSensor * device);
static void MicroclimateSensor_startCheckParametersTimer(
        MicroclimateSensor * device);
static void MicroclimateSensor_stopCheckParametersTimer(
        MicroclimateSensor * device);
static void MicroclimateSensor_initializeDefault(MicroclimateSensor *device);
static void MicroclimateSensor_handleCurrentParameters(
        MicroclimateSensor *device, Sensor_Results * results);
static Sensor_Results MicroclimateSensor_computeAverage(
        MicroclimateSensor * device);
static void MicroclimateSensor_updateMeasurementPointsCountForAverage(
        MicroclimateSensor *device);
static void scheduleItemChangedCallback(ScheduleItemBase *scheduleItem,
                                        void *arg);

static ScheduleItemBase* retrieveScheduleItemBaseFunction(void *items,
                                                  uint16_t itemIndex);

static void * checkParametersTask(void *arg);


void MicroclimateSensor_turnedOff(MicroclimateSensor * device)
{
    MicroclimateSensor_stopCheckParametersTimer(device);
    device->state = DISABLED;
}
void MicroclimateSensor_turnedOn(MicroclimateSensor * device)
{
    device->state = true;
    MicroclimateSensor_startCheckParametersTimer(device);
}

void MicroclimateSensor_startCheckParametersTimer(MicroclimateSensor * device)
{
    Utils_setupTimer(&device->checkParametersSemaphore,
                     &device->checkParametersClock, 0,
                     device->measurePeriodInMilliseconds * 1000000, true);
```

```
}

void MicroclimateSensor_stopCheckParametersTimer(MicroclimateSensor * device)
{
   Utils_cancelTimer(&device->checkParametersClock);
}



static void MicroclimateSensor_handleCurrentParameters(
      MicroclimateSensor *device, Sensor_Results * results)
{
   EventHandle event = { 0 };
   int8_t currentHumidityComparisionWithLowerBoundary = compareFloat(
         results->humidity, device->parameters.humidityLowerBoundary,
         DEFAULT_FLOAT_COMPARE_PRECISION);
   int8_t currentHumidityComparisionWithUpperBoundary = compareFloat(
         results->humidity, device->parameters.humidityUpperBoundary,
         DEFAULT_FLOAT_COMPARE_PRECISION);
   int8_t currentTemperatureComparisionWithLowerBoundary = compareFloat(
         results->temperature, device->parameters.temperatureLowerBoundary,
         DEFAULT_FLOAT_COMPARE_PRECISION);
   int8_t currentTemperatureComparisionWithUpperBoundary = compareFloat(
         results->temperature, device->parameters.temperatureUpperBoundary,
         DEFAULT_FLOAT_COMPARE_PRECISION);

   if (currentHumidityComparisionWithLowerBoundary <= 0)
   {
      float difference = device->parameters.humidityLowerBoundary
            - results->humidity;
      event.arg = &difference;
      if (currentHumidityComparisionWithLowerBoundary == 0)
      {
         event.eventId = HUMIDITY_REACHED_THE_LOWER_BOUNDARY;
      }
      else
      {
```

```
        event.eventId = HUMIDITY_CROSSED_THE_LOWER_BOUNDARY;
    }
}
else if (currentHumidityComparisionWithUpperBoundary >= 0)
{

    float difference = results->humidity
            - device->parameters.humidityUpperBoundary;
    event.arg = &difference;
    if (currentHumidityComparisionWithUpperBoundary == 0)
    {
        event.eventId = HUMIDITY_REACHED_THE_UPPER_BOUNDARY;
    }
    else
    {
        event.eventId = HUMIDITY_CROSSED_THE_UPPER_BOUNDARY;
    }
}
else if (currentTemperatureComparisionWithLowerBoundary <= 0)
{
    float difference = device->parameters.temperatureLowerBoundary
            - results->temperature;
    event.arg = &difference;

    if (currentTemperatureComparisionWithLowerBoundary == 0)
    {
        event.eventId = TEMPERATURE_REACHED_THE_LOWER_BOUNDARY;
    }
    else
    {
        event.eventId = TEMPERATURE_CROSSED_THE_LOWER_BOUNDARY;
    }
}
else if (currentTemperatureComparisionWithUpperBoundary >= 0)
{
    float difference = results->temperature
            - device->parameters.temperatureUpperBoundary;
```

```
        event.arg = &difference;

        if (currentTemperatureComparisionWithUpperBoundary == 0)
        {
            event.eventId = TEMPERATURE_REACHED_THE_UPPER_BOUNDARY;
        }
        else
        {
            event.eventId = TEMPERATURE_CROSSED_THE_UPPER_BOUNDARY;
        }
    }
    if (event.arg != NULL)
    {
        MicroclimateSensor_onEvent(device, event);
    }
}
static Sensor_Results MicroclimateSensor_computeAverage(
        MicroclimateSensor * device)
{
    uint16_t i;
    Sensor_Results averageResults = { 0 };
    for (i = 0; i < device->averageMeasurementPointsCount; i++)
    {
        averageResults.humidity +=
                device->measurementsInComputeAveragePeriod[i].humidity;
        averageResults.temperature +=
                device->measurementsInComputeAveragePeriod[i].temperature;
    }
    averageResults.humidity /= device->averageMeasurementPointsCount;
    averageResults.temperature /= device->averageMeasurementPointsCount;
    return averageResults;
}
void *checkParametersTask(void *arg)
{
    MicroclimateSensor * device = arg;
    HDC1080_Sensor * sensor = device->microclimateSensor;
    sem_t * checkSemaphore = &device->checkParametersSemaphore;
```

```
    Sensor_Results results;
    while (1)
{
        sem_wait(checkSemaphore);
        results = HDC1080_getTemperatureAndHumidity(sensor);
        device->measurementsInComputeAveragePeriod[device->measuredPointsCount] =
                results;
        MicroclimateSensor_handleCurrentParameters(
                device,
                &device->measurementsInComputeAveragePeriod[device->measuredPointsCount]);
        device->measuredPointsCount++;
        if (device->measuredPointsCount
                == device->averageMeasurementPointsCount)
        {
            Sensor_Results averageAtPeriod = MicroclimateSensor_computeAverage(
                    device);
            EventHandle event = {
                    .eventId = COMPUTED_AVERAGE_AT_SELECTED_PERIOD, .arg =
                            &averageAtPeriod };
            MicroclimateSensor_onEvent(device, event);
        }
}
}

MicroclimateSensor * MicroclimateSensor_init(
        HDC1080_Sensor *temperatureAndHumiditySensor)
{
    MicroclimateSensor * microclimateSensor = (MicroclimateSensor *) calloc(
            1, sizeof(MicroclimateSensor));
    MicroclimateSensor_initializeDefault(microclimateSensor);
    microclimateSensor->microclimateSensor = temperatureAndHumiditySensor;
    sem_init(&microclimateSensor->checkParametersSemaphore, 0, 0);
    MicroclimateSensor_configure(microclimateSensor);
    return microclimateSensor;

}
void MicroclimateSensor_configure(MicroclimateSensor *device)
```

```
{

   Utils_createTask(CHECK_PARAMETERS_TASK_STACK_SIZE,
   CHECK_PARAMETERS_TASK_PRIORITY,
                checkParametersTask,
                device);
   MicroclimateSensor_turnedOn(device);
}
bool MicroclimateSensor_setState(MicroclimateSensor *device,
bool state)
{
if (state)
{
      MicroclimateSensor_turnedOn(device);
}
else
{
      MicroclimateSensor_turnedOff(device);
}
   return true;
}
DeviceState MicroclimateSensor_getState(MicroclimateSensor *device)
{
return device->state;
}

void MicroclimateSensor_setCurrentTime(MicroclimateSensor *device,
                              Time currentTime)
{
   if (device->schedule != NULL)
{
      Schedule_updateCurrentTime(device->schedule, currentTime);
   }
   else
   {
   device->schedule = Schedule_init(currentTime, scheduleItemChangedCallback,
                           device, retrieveScheduleItemBaseFunction);
```

```
    }
}
void MicroclimateSensor_notifyAboutEvent(MicroclimateSensor *device,
                                         EventHandle event)
{

}


void MicroclimateSensor_onEvent(MicroclimateSensor *device,
                                EventHandle eventHandle)
{
uint8_t reaction;
for (reaction = 0; reaction < REACTIONS_COUNT; reaction++)
{
if (device->eventsReactions[eventHandle.eventId][reaction])
{
        MicroclimateSensor_realizeReaction(device,
                                           (DeviceReaction) reaction,
                                           eventHandle);
}
}

}
void MicroclimateSensor_realizeReaction(MicroclimateSensor *device,
                                        DeviceReaction reaction,
                                        EventHandle event)
{
switch (reaction)
{
case NOTIFY_CENTER:
      MicroclimateSensor_notifyAboutEvent(device, event);
break;
   case ASK_TO_COMPENSATE_THE_TEMPERATURE_DIFFERENCE:

      MicroclimateSensor_askForCompensatingTemperatureDifference(
            device,
```

```
                                    *((float *) event.arg));
break;
   case ASK_TO_COMPENSATE_THE_HUMIDITY_DIFFERENCE:
      MicroclimateSensor_askForCompensatingHumidityDifference(
            device, *((float *) event.arg));
      break;
}
}
static void MicroclimateSensor_initializeDefault(MicroclimateSensor *device)
{
   device->measurePeriodInMilliseconds = MEASURE_TIME_IN_MS;
   device->averageComputingPeriod = DEFAULT_AVERAGE_MEASUREMENT_TIME_INTERVAL;
   MicroclimateSensor_updateMeasurementPointsCountForAverage(device);
   MicroclimateSensor_startCheckParametersTimer(device);
}
static void MicroclimateSensor_updateMeasurementPointsCountForAverage(
      MicroclimateSensor *device)
{
   device->averageMeasurementPointsCount = device->averageComputingPeriod
         / device->measurePeriodInMilliseconds;
   if (device->measurementsInComputeAveragePeriod != NULL)
   {
      free(device->measurementsInComputeAveragePeriod);
   }
   device->measurementsInComputeAveragePeriod = malloc(
         device->averageMeasurementPointsCount * sizeof(Sensor_Results));
}
void MicroclimateSensor_setMeasurePeriod(MicroclimateSensor *device,
                                 uint32_t period)
{
   MicroclimateSensor_stopCheckParametersTimer(device);
   device->measurePeriodInMilliseconds = period;
   MicroclimateSensor_updateMeasurementPointsCountForAverage(device);
   MicroclimateSensor_startCheckParametersTimer(device);

}
void MicroclimateSensor_setComputingAveragePeriod(MicroclimateSensor *device,
```

```
                                    uint32_t period)
{
   MicroclimateSensor_stopCheckParametersTimer(device);
   device->averageComputingPeriod = period;
   MicroclimateSensor_updateMeasurementPointsCountForAverage(device);
   MicroclimateSensor_startCheckParametersTimer(device);
}
void scheduleItemChangedCallback(ScheduleItemBase *scheduleItem, void *arg)
{
MicroclimateSensor *sensor = arg;
MicroclimateScheduleItem *item = (MicroclimateScheduleItem *) scheduleItem;
MicroclimateSensor_setState(sensor, item->base.isEnabled);
MicroclimateSensor_updateParameters(sensor, item->parameters);
}

ScheduleItemBase* retrieveScheduleItemBaseFunction(void *items,
                                    uint16_t itemIndex)
{
MicroclimateScheduleItem *scheduleItems = items;
return &scheduleItems[itemIndex].base;

}

void MicroclimateSensor_updateParameters(
   MicroclimateSensor *device, MicroclimateSensorParameters parameters)
{
device->parameters = parameters;
}
void MicroclimateSensor_setDaySchedule(MicroclimateSensor *device,
                            DaySchedule *schedule, uint8_t weekDay)
{
Schedule_setDaySchedule(device->schedule, weekDay, schedule,
                    sizeof(MicroclimateScheduleItem));

}
void MicroclimateSensor_setWeekSchedule(MicroclimateSensor *device,
                             WeekSchedule schedule)
```

```
{
Schedule_setWeekSchedule(device->schedule, schedule,
                    sizeof(MicroclimateScheduleItem));
}
```

```
//BaseSmartHouseDeviceOperations.java
package com.dozy.devices.smarthouse.common.operations;


import com.dozy.devices.smarthouse.common.dto.SmartHouseDeviceDto;
import com.dozy.libraries.common.service.operations.BaseCrudOperations;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;
import java.util.Optional;

@RequestMapping("/devices")
public interface BaseSmartHouseDeviceOperations<D extends SmartHouseDeviceDto> extends
BaseCrudOperations<D> {
   @GetMapping("/room/{roomId}")
   Optional<List<D>> getDevicesByRoomId(@PathVariable Long roomId);

   @GetMapping("/{id}/root")
   Optional<Long> getRootDevice(@PathVariable Long id);
}

//BaseDeviceConfigurationOperations.java
package com.dozy.devices.smarthouse.common.operations;

import com.dozy.devices.smarthouse.common.dto.SmartHouseDeviceConfigurationDto;
import com.dozy.libraries.common.service.operations.BaseCrudOperations;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import java.util.List;
import java.util.Optional;

@RequestMapping("/configurations")
public interface BaseDeviceConfigurationOperations<
        O extends SmartHouseDeviceConfigurationDto> extends BaseCrudOperations<O> {
    @GetMapping("/user/current/all")
    Optional<List<O>> getAllAvailableUserDefinedSmartHouseDeviceConfigurations();
}
```

ДОДАТОК 3

Розробка мультиагентної бездротової Mesh – мережі Smart пристроїв

Опис програми

НТУУ «КПІ» ТЕФ АПЕПС ТІ-61

Листів 4

АНОТАЦІЯ

Розроблений програмно-апаратний комплекс забезпечує керування та моніторинг параметрів навколишнього середовища. Дипломна робота присвячена розробці мультиагентної системи моніторингу та управління на базі CC1310, зпсобами побудови мікросервісної архітектури, Dart і Flutter

ЗМІСТ

1. ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

Клієнт-додаток називається DozySmartHouse. Даний продукт розорблено у середовищі Android Studio 3.0, використовуючи мову програмування Dart, фреймворк Flutter. Прошивка мікроконтролеру була розроблена на мові С, серверна частина на мові програмування Java. Програма призначена для контролю і моніторингу бездротової мультиагентної мережі.

1.1. Опис логічної структури

Програмний продукт мобільного застосунку було розроблено у вигляді мультиплатформенного додатку за допомогою Flutter.

Данна система складається з 3 основних частин: прошивка мікроконтролерів, серверний додаток та клієнт-додаток. Прошивка мікроконтролерів складається з окремих логічних блоків для збирання кожного типу даних. Серверний додаток отримує, аналізує інформацію від мультиагентної системи, а також приймає запити від мобільного додатку.

1.2. Вхідні та вихідні дані

Вихідними даними для контролеру є дані з датчиків та значення,

Вхідними даними для контролеру є конфігурація пристрою, надіслана з серверного додатку.

Вхідними даними для серверного додатку є дані з контролеру та команди з клієнт-додатку.

Вихідними даними є команди для мультиагентної системи та поточні значення системи для клієнт-додатку.

Вхідними даними для мобільного додатку є дані з серверного додатку та команди від користувача.

Вихідними даними для клієнт додатку є команди, введені користувачем.